МОНИТОР

журнал для программистов — практиков

8.92

ISSN 0869-3889

PROGRAMMER

Шифрование: системы с открытым ключом

Имеющий уши да услышит!

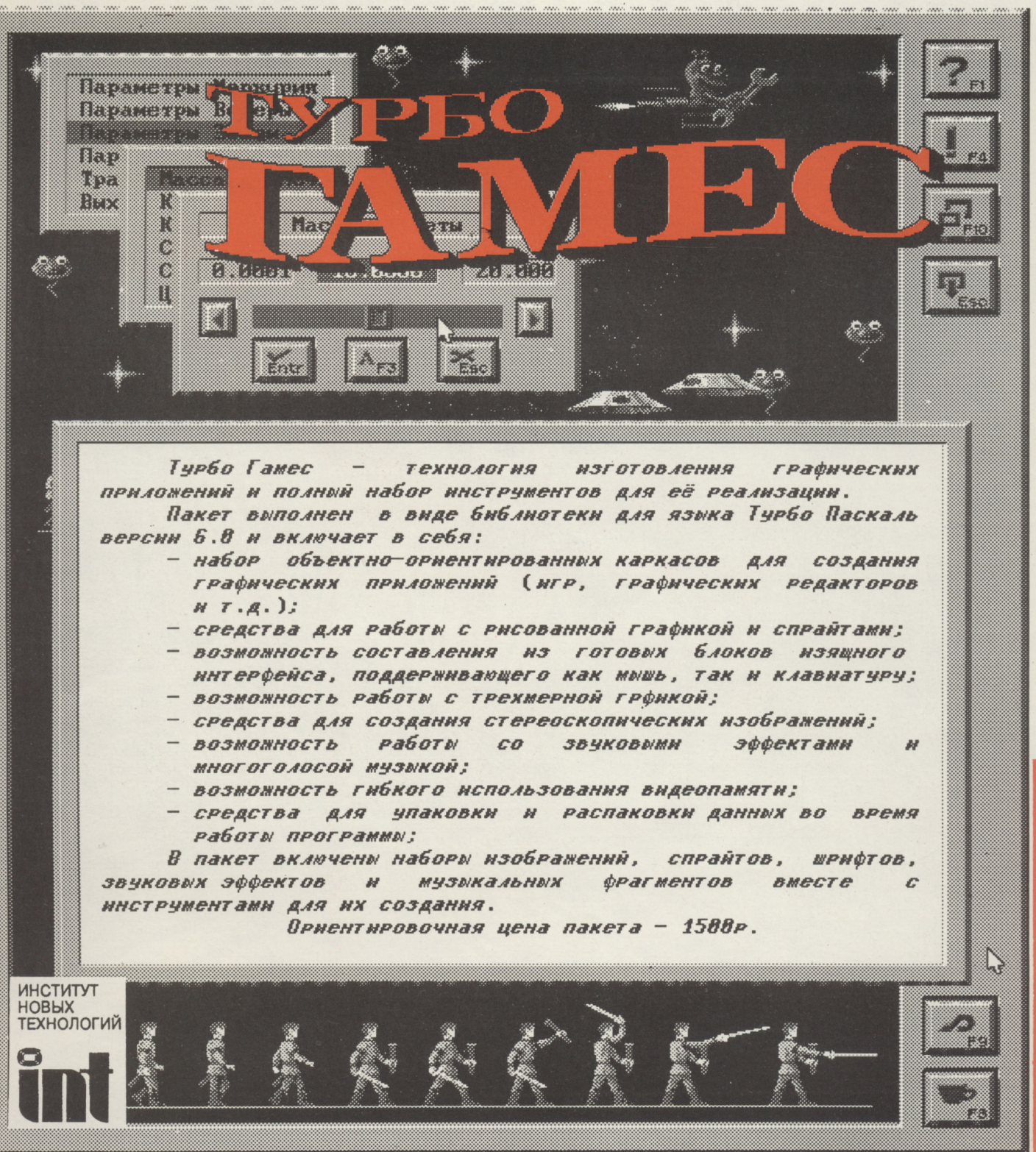
Укрощение строптивого или двадцать четыре вместо девяти

USER

Где взять миллиард?

Приключенческие игры: попытка осмысления жанра

Бесконечны комбинации модулей ТУРБО ГАМЕС, а создать игру Вы сможете за 1 день!



Турбо Гамес - технология изготовления графических приложений и полный набор инструментов для её реализации.

Пакет выполнен в виде библиотеки для языка Турбо Паскаль версии 6.0 и включает в себя:

- набор объектно-ориентированных каркасов для создания графических приложений (игр, графических редакторов и т.д.);
- средства для работы с рисованной графикой и спрайтами;
- возможность составления из готовых блоков изящного интерфейса, поддерживающего как мышь, так и клавиатуру;
- возможность работы с трехмерной графикой;
- средства для создания стереоскопических изображений;
- возможность работы со звуковыми эффектами и многоголосой музыкой;
- возможность гибкого использования видеопамяти;
- средства для упаковки и распаковки данных во время работы программы;

В пакет включены наборы изображений, спрайтов, шрифтов, звуковых эффектов и музыкальных фрагментов вместе с инструментами для их создания.

Ориентировочная цена пакета - 1500р.

ИНСТИТУТ НОВЫХ ТЕХНОЛОГИЙ

int

Подробную информацию о ТУРБО ГАМЕС Вы можете получить по телефону:
(095) 238-99-33.
Факс: (095) 238-32-78.

МОНИТОР

Уважаемый читатель!

В руках у Вас последний, восьмой, номер "Монитора" за 1992 год. Я пишу эти строки 7 декабря 1992 года; читаете Вы их уже в наступившем 1993 году. Конец года - самое время для подведения итогов и построения планов на будущее.

Вследствие молодости как самого журнала (самый первый номер "Монитора" вышел год тому назад - в декабре 1991 года), так и нашей редакции (средний возраст - 29 лет), вспоминать минувшее нам еще рано, да, впрочем, и скучно. Поговорим лучше о будущем, а именно - о том, что ждет читателей "Монитора" в 1993 году.

Пролистав несколько последних номеров "Монитора", Вы обнаружите, что все больше и больше статей мы, несмотря на отчаянные протесты художника, набирали мелким шрифтом. Это объяснялось нашим стремлением вместить в ограниченный объем журнала как можно больше интересных для нашего читателя статей. В 1993 году "Монитор" станет толще - 80 страниц, выходить будет чаще - 10 номеров, поэтому читать Вам станет легче - мы вернемся к нормальному шрифту.

Остаются неизменными ориентация журнала на программистов-практиков и набор наших основных тем: технологии программирования, базы данных, графика, работа на аппаратном уровне, языки программирования, компиляторы. Большая часть статей будут содержать листинги программ и/или алгоритмы.

С точки зрения языковой мы по-прежнему будем делить журнальную площадь между приверженцами Си, Паскаля и Ассемблера. Любителей Бейсика должен привлечь цикл статей о профессиональном программировании в системе QuickBasic.

Для тех наших читателей, которые уже перешли на работу в Windows или намереваются сделать это, мы открываем постоянную рубрику, посвященную программированию в среде Windows. Впрочем, мы не намерены обходить вниманием OS/2, Unix, QNX, так же как и замыкаться лишь на PC - в 1993 году у нас откроется рубрика "Рабочие станции".

В рубрике САПР мы будем продолжать публиковать статьи по программированию в среде Автокад.

Для поклонников компьютерных игр в каждом номере журнала - экспертный анализ наиболее интересных игр и тенденций развития жанра, проводимый специалистами из фирмы "Инфорком".

Если предложенная программа Вас устраивает, то подписной купон Вы найдете на стр. 17.

С Новым годом!

Журнал для программистов-практиков

Редколлегия:
Д.Э.Бройтман (главный редактор)
А.А.Винокуров (отдел распространения)
В.В.Водолазский
Е.В.Гун
С.В.Макельский (главный художник)
А.М.Миляков (ответственный секретарь)
Н.А.Никитина (отдел рекламы)
А.В.Орехов (издательский директор)
А.В.Семенников (компьютерная верстка)
Л.И.Турок

Представительство в Европе (Германия):
Марина Гетманская
тел. 8-(0631)-7-49-05

В оформлении номера использованы рисунки
К.З.Гусалова и А.С.Замулы

Подписано в печать с оригинал-макета 11.12.92.
Формат 60x90/8.
Тираж 30 000 экз.
Печать офсетная. Печ.л.8.0.
Изд. № 1971.
Цена договорная
Зак. 185
Ф-ка "Картолитография"

Издатель:
Научно-техническое общество
Софт·Москва

Для писем: 117419, г.Москва,а/я 765
Тел. (095)238-99-33
Факс (095)238-32-78

© "Монитор", 1992

Базы данных

С. Кравивенко. **Clipper или Paradox: попытка аргументированного выбора Часть 4** 4

В статье заканчивается обсуждение сильных и слабых сторон двух распространенных СУБД, приводятся примеры программ.

Л. Ливеровский. **Модельный файл Clarion** 10

Технологии программирования

С. Трепыхалин. **Несколько полезных макросов для Turbo Pascal 6.0** 14

Д. Солдатенков. **Имеющий уши да услышит** 16

Создание звука на IBM PC.

Защита программ

В. Водолазкий. **Коммерческие системы шифрования: основные алгоритмы и их реализация Часть 2. Системы с открытым ключом** 19

Обсуждаются принципы построения систем с открытым ключом. Детально описаны система распределения ключей Диффи-Хелмана и алгоритм RSA.

Бизнес

Д. Фивейский. **От программиста до конечного пользователя: программные продукты Computer Associates** 23

Компьютер в разрезе

М. Иринчук. **Проблема памяти и адаптер VGA** 24

В статье рассматривается проблема использования видеопамяти для хранения данных.

Д. Правиков. **Замедление работы программы** 26

В статье описаны три способа искусственного замедления работы программ.

Н. Косарев. **Укрощение строптивого, или двадцать четыре вместо девяти** 28

Вторая из серии статей, посвященных возможности копирования экрана дисплея на разные типы принтеров.

Игры

Adventure Games: эволюция жанра 31

Г. Шабат. **Почему я лютый враг компьютерных игр** 35

Сети

А. Ермолов. **Что такое IPX и как его использовать в своих целях Часть 2** 36

Продолжение описания сетевого протокола IPX. Приводятся примеры различных программ, использующих вызовы его пользовательского интерфейса, в том числе резидентная программа приема сообщений.

Операционные системы

Ю. Крючков. **"Вскрытие" парольной защиты в операционных системах DR DOS** 37

Электронные таблицы

Ю. Крючков. **Размышления об отличиях в расчетах дисперсии и среднекватратичного отклонения в SuperCalc** 39

Языки программирования

А. Сидоров. **Оптимизация в Zortech C++** 42

Сравнительный анализ качества оптимизации наиболее распространенных компиляторов с языка С.

А. Кельманзон. **Особенности C++ в компиляторах Borland** 49

В статье описаны особенности программирования на языке C++ в Borland C++ версии 2.0 и 3.0, связанные с появлением в языке новых синтаксических конструкций; там, где это возможно, рассказывается о способах их реализации.

Этюды для программистов

М. Трофимов. **Итерация против рекурсии: проблема выбора** 53

На примере ряда классических задач программирования обсуждается круг вопросов, связанных с применением рекурсии в программировании на языке Турбо Паскаль. Приводится сравнение с нерекурсивной реализацией тех же алгоритмов.

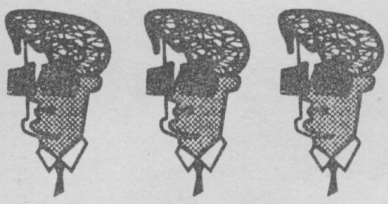
Суперкомпьютеры

К. Злобин. **Где взять миллиард ?** 60

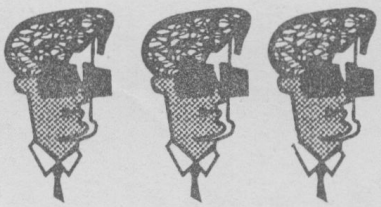
Статья посвящена ЭВМ сверхвысокой производительности. Описаны принципы векторно-конвейерной архитектуры. Рассмотрены примеры зарубежных и отечественных машин этого класса.

Clipper или Paradox : попытка аргументированного выбора

Часть 4



С. Крапивенко



4.10. Реляционные команды

К реляционным командам относятся те, что работают с наборами записей (таблицами) целиком, в отличие от команд, оперирующих с понятием "текущая запись". Как правило, использование реляционных команд предпочтительнее, ввиду их большей эффективности.

4.10.1. Реляционные команды Clipper

Clipper содержит ряд команд, работающих как со всеми записями таблицы (то есть реляционный вариант), так и с группой записей, в том числе с одной (текущей записью). В зависимости от команды и определенных ключевых слов, предполагается либо текущая запись, либо группа записей после текущей, либо группа записей, удовлетворяющих условию, либо все записи. Поясним синтаксис реляционных команд Clipper, приведенных в таблице 10.

Таблица 10. Реляционные команды Clipper

APPEND [<список полей>] FROM <таблица-источник> [<условие>] добавление записей из таблицы-источника в текущую таблицу
AVERAGE [<зона действия>] <список полей> TO <список переменных> [<условие>] подсчет средних значений (матожиданий) полей с сохранением результатов в переменных списка
COPY TO <таблица-результат> [<зона действия>] [FIELDS <список полей>] [<условие>] копирование записей из текущей таблицы в результирующую; если не указан список полей, копируются все поля
COUNT [<зона действия>] [<условие>] TO <переменная> подсчет записей текущей таблицы с сохранением результата в переменной
DELETE [<зона действия>] [<условие>] помечает записи как удаленные; при отсутствии зоны действия и условия относится только к текущей записи; записи могут быть восстановлены командой RECALL с аналогичными параметрами; физическое удаление осуществляется командой PACK
JOIN WITH <область-источник> TO <таблица-результат> FOR <условие добавления> FIELDS <список полей двух таблиц> сформировать новую таблицу как результат объединения полей из одной и другой таблицы в соответствии с условием
REPLACE [<зона действия>] <поле1> WITH <выражение1> [,<поле2> WITH <выражение2>...] [<условие>] заменить значения полей выражениями; при отсутствии зоны действия и условия относится только к текущей записи
SUM [<зона действия>] <список полей> TO <список переменных> [<условие>] подсчет сумм значений полей с сохранением результатов в переменных списка
TOTAL ON <ключевое поле> TO <таблица-результат> [<зона действия>] [FIELDS <список полей>] [<условие>] подсчет сумм и формирование итоговых записей в новой таблице для групп записей с одинаковым значением ключевого поля; в суммировании участвуют поля из списка или все числовые поля, если список не указан
UPDATE ON <ключевое поле> FROM <область-источник> [RANDOM] REPLACE <поле1> WITH <выражение1> [,<поле2> WITH <выражение2>...] замена в текущей таблице полей выражениями на основе записей источника; обе таблицы должны быть проиндексированными по ключевому полю, если не указано слово [RANDOM] - в этом случае записи источника рассматриваются в произвольном порядке

Квадратные скобки указывают на необязательность параметра.

Зона действия определяет, на какие записи действует команда. Возможны следующие значения: ALL - вся таблица, REST - записи от текущей записи до конца таблицы, NEXT <n> - n записей, включая текущую. При отсутствии зоны в командах TOTAL, SUM и AVERAGE подразумевается ALL, а в командах REPLACE и DELETE - NEXT 1, то есть текущая запись.

Условие возможно двух типов, с ключевыми словами FOR и WHILE. Первый вариант (FOR) аналогичен фильтру и действует на все записи. Второй вариант (WHILE) позволяет обработать все подряд идущие записи, для которых истинно ограничивающее условие. Если указано условие, зона действия определения не требует.

Список полей - это имена полей через запятую.

Список переменных - имена переменных через запятую.

Необходимо заметить, что если в качестве источника указано слово "таблица", то эта таблица должна быть закрыта, а если "область", то открыта; если в качестве результата выступает таблица, то таблицы с таким именем быть не должно, в противном случае команды могут вести себя в стиле плохой шутки (например, TOTAL закроет исходную таблицу без выдачи сообщения, а выполнение программы продолжится).

4.10.2. Реляционные средства Paradox

По мощности и составу реляционных средств Paradox значительно превосходит аналогичные средства Clipper (табл. 11 - 13).

Заметим, что для вызова команд Add, MultiAdd, FormAdd и Subtract необходима следующая последовательность команд (клавиш): MENU {Tools} {More}, затем идентификатор команды в фигурных скобках и далее требуемые имена таблиц и параметры. Кроме того, эти команды требуют главного режима Paradox (Main), что, в свою очередь, несовместимо с режимом корректировки таблиц или конструирования экранных или печатных форм, графов и т.п. Напомню, что в Paradox для входа в корректировку таблицы требуется команда EditKey или CoEditKey (соответственно, клавиши F9 и Alt-F9), а при завершении

Таблица 11. Реляционные комады Paradox

{Add} <таблица-источник> <таблица-результат> {NewEntries} добавить записи из одной таблицы к другой той же структуры
{Add} <таблица-источник> <таблица-результат> {Update} использовать записи одной таблицы для замены записей в другой
{MultiAdd} <таблица-источник> <таблица-список> {NewEntries} добавить записи из одной таблицы в две или более таблиц той же структуры; список таблиц задан в таблице (второй параметр)
{MultiAdd} <таблица-источник> <таблица-список> {Update} использовать записи одной таблицы для замены записей в двух или более таблицах той же структуры; список таблиц задан в таблице (второй параметр)
{FormAdd} <таблица-результат> <экранная форма> {EntryTables} добавить записи из одной или нескольких Entry таблиц (таблиц новых записей) к целевой таблице
{FormAdd} <таблица-результат> <экранная форма> {AnyTables} {NewEntries} <таблица-источник> добавить записи из одной таблицы к другой; состав полей для копирования взять из экранной формы
{FormAdd} <таблица-результат> <экранная форма> {AnyTables} {Update} <таблица-источник> использовать записи одной таблицы для замены записей в другой; состав полей для замены взять из экранной формы
{Subtract} <таблица-источник> <таблица-результат> удаление из таблицы-результата записей, совпавших с записями таблицы-источника
{Ask} <таблица> включить указанную таблицу в запрос; для ввода запроса формируется таблица той же структуры, в которую вводятся предложения языка запроса; таблиц в запросе может быть несколько; результат запроса - таблица ANSWER

- либо подтверждение изменений - Do_it! (F2), либо отказ (для режима Edit - MENU {Cancel} {Yes}). В программе выбор альтернативы из меню можно сделать функцией SELECT (<значение>). Для входа в режим запроса последовательность команд следующая: MENU {Ask} <имя таблицы>.

Записи с дублированным индексным ключом, возникшие при выполнении команд добавления или модификации записей, направляются в таблицу с именем KEYVIOL, замененные записи - в таблицу CHANGED; если ввод осуществлялся из специальных таблиц ввода (Entry Tables), то записи, вызывающие проблемы, остаются там же. Если возникли проблемы, а рабочий и личный каталог не совпадают, также формируется таблица LIST из двух полей, содержащих полные пути к "проблемным" временным таблицам и таблицам, с записями которых возник конфликт.

Статистические функции (табл. 12), в отличие от Clipper, позволяют обращаться даже к неоткрытым таблицам, а функции с префиксом "Image" выбирают только те записи связанных таблиц, которые удовлетворяют текущему типу множественных связей (1:1, 1:M, M:1, M:M), заданных текущей экранной формой.

Таблица 12. Статистические функции Paradox

CAVERAGE (<таблица>, <поле>)	среднее значение для столбца таблицы
CCOUNT (<таблица>, <поле>)	подсчет непустых значений в столбце
CMAX (<таблица>, <поле>)	наибольшее значение в столбце таблицы
CMIN (<таблица>, <поле>)	наименьшее значение в столбце таблицы
CSTD (<таблица>, <поле>)	стандартное отклонение в столбце
CSUM (<таблица>, <поле>)	подсчет суммы значений в столбце
CVAR (<таблица>, <поле>)	сумма отклонений от среднего в столбце
ImageCAverage ()	среднее для связанных записей
ImageCCount ()	число связанных записей
ImageCMax ()	наибольшее значение среди связанных записей
ImageCMin ()	наименьшее значение среди связанных записей

5. Программирование

В этом разделе рассматриваются некоторые элементы языков программирования Paradox и Clipper, не встречавшиеся в предыдущих разделах и являющиеся, по мнению автора, важными при разработке и отладке программ. Поскольку рамки статьи ограничены, для получения более полной информации обращайтесь к документации.

Несколько общих замечаний.

Для выполнения в среде Paradox программа на языке PAL должна находиться либо в текстовом файле с расширением SC

Таблица 13. Режим запросов Paradox (ASK/QUERY)

Клавиши [1]:
CHECK (F6) - включение поля в ответ; сортировка записей по возрастанию значения поля; из ответа исключаются дублированные записи;
CHECKPLUS (Alt-F6) - включение поля в ответ; без сортировки и удаления совпадающих записей;
CHECKDESCENDING (Ctrl-F6) - включить поле в ответ; сортировка записей по убыванию значения поля с исключением дублированных записей;
GROUPBY (Shift-F6) - определение набора записей для операторов работы с набором; ставится в крайнем левом поле;
[2] (F5) - после нажатия этой клавиши до ближайшего пробела вводимый текст рассматривается как переменная, ассоциированная со значением данного поля, либо как метка для организации связей между таблицами; в последнем случае та же метка должна быть помещена в связываемую таблицу запроса в ассоциированное поле; переменные могут быть использованы в ограничивающих выражениях или при вычислении новых полей;
DO_IT! (F2) - выполнение запроса.
Изменение текущей таблицы [3]:
INSERT - вставка новых записей;
DELETE - удаление записей;
CHANGETO - замена на новые значения;
FIND - копирование записей в таблицу FINDED.
Определение и использование набора записей:
SET [2] - определение выбранных записей в качестве набора записей;
ONLY - выбор записей, соответствующих единственному члену набора;
NO - выбор записей, не соответствующих ни одному члену набора;
EVERY - выбор записей, соответствующих каждому члену набора;
EXACTLY - группа выбранных записей должна полностью соответствовать набору.
Вычисление новых полей [4]:
CALC - вычисление нового поля; за опцией CALC следует один из следующих операторов:
SUM - сумма значений;
AVERAGE - среднее из значений;
MAX - наибольшее значение;
MIN - наименьшее значение;
COUNT - число значений;
Операторы сравнения: = > < >= <=
Арифметические операторы: + - * / ()
Образцы для поиска:
LIKE - неточное соответствие;
.. - любая группа символов;
@ - один любой символ;
TODAY - текущий день;
BLANK - пустое поле;
NOT - отрицание.
Специальные операторы:
OR - логическое ИЛИ;
, - логическое И;
AS <имя> - изменение имени текущего поля;
! - включение в ответ всех записей.
[1] клавиши употребляются в интерактивном режиме запросов (ASK), в тексте программы (оператор QUERY .. ENDQUERY) - их аналоги;
[2] в QUERY переменные определяются первым символом "_" (нижнее подчеркивание);
[3] опции ставятся в самом левом поле строки запроса;
[4] имя поля в ответе формируется из ключевого слова, слова " of " и имени исходного поля, например, "Sum of Стоимость".

(такие файлы называются программами на языке PAL), либо в библиотечных файлах (расширение LIB), содержащих оттранслированный код процедур. Для запуска программы на выполнение существует команда PLAY <имя файла> или это можно сделать из меню (MENU {Scripts} {Play} select (name)). При старте Paradox в командной строке могут быть указаны имя программы для выполнения (в этом случае не выдается заставка Paradox) и несколько параметров, в том числе установка минимального размера буферов для таблиц (от 24 Кбайтов), имени пользователя, пути к сетевому каталогу, защищенного или реального режима процессора и т.д.. Если в текущем каталоге присутствует файл INIT.sc, он стартует автоматически.

Для выполнения программы на языке Clipper необходимы следующие шаги: подготовка исходных текстов программ в файлах

с расширением PRG (при необходимости еще и файлов экранных форм с расширением FMT), трансляция PRG файлов компилятором (CLIPPER <имя файла>), связывание объектных файлов и библиотек (PLINK86 FI <список объектных файлов> LIB <список библиотек>) и собственно запуск полученного EXE файла. Главная программа должна быть первой в списке для компоновки. В списке библиотек должны быть указаны библиотеки CLIPPER.LIB и EXTEND.LIB. Для оверлейных программ требуется дополнительно библиотека OVERLAY.LIB. Если в приложении открывается одновременно более 20 таблиц, индексных и прочих файлов, в файл AUTOEXEC.BAT внесите установку

SET CLIPPER=F<n>,

где <n> - требуемое число плюс 5,
и это же значение - в файл CONFIG.SYS:

FILES=<n> и BUFFERS=<m>,

где <m> - число таблиц, индексных и прочих файлов.

В версии 5 поддерживаются DOS установки SET LIB, SET INCLUDE и др. Кроме того, в версии 5 имя компоновщика - RTLINK.

5.1. Управляющие структуры

Оба языка - PAL и Clipper - включают достаточный набор управляющих конструкций для структурного программирования (табл. 14).

Таблица 14. Операторы управления

Назначение	Paradox	Clipper
Оператор условия	IF <условие> <операторы> [ELSE <операторы>] ENDIF	IF <условие1> <операторы> [ELSEIF <условие2> <операторы>] [ELSE <операторы>] ENDIF или IIF (<условие>,<выражение1>,<выражение2>)
Оператор выбора	SWITCH CASE <условие> : <операторы> ... [OTHERWISE: <операторы>] ENDSWITCH	DO CASE CASE <условие> <операторы> ... [OTHERWISE: <операторы>] ENDCASE
Цикл с шагом	FOR <переменная> FROM <n1> TO <n2> [STEP <n3>] <операторы> ENDFOR	FOR <переменная> = <n1> TO <n2> [STEP <n3>] <операторы> NEXT
Цикл по условию	WHILE <условие> <операторы> ENDWHILE	DO WHILE <условие> <операторы> ENDDO
Цикл по всей таблице	SCAN [FOR <условие>] <операторы> ENDSCAN	Отсутствует
Продолжить цикл	LOOP	LOOP
Выйти из цикла	QUITLOOP	EXIT
Обработка исключительных ситуаций	Отсутствует	BEGIN SEQUENCE <операторы> END
Переход на конец обработки исключительных ситуаций	Отсутствует	BREAK

В выражениях логического типа могут быть использованы логические константы (TRUE / FALSE в Paradox и .T. / .F. в Clipper) и логические операции - И (AND / .AND.), ИЛИ (OR / .OR.), НЕ (NOT / .NOT.).

Полезная конструкция

IIF (<условие>,<выражение 1>,<выражение 2>)

языка Clipper используется в выражениях для подстановки: выражение 1, если условие истинно, и выражение 2 - в противоположном случае.

Оператор обработки исключительных ситуаций

BEGIN SEQUENCE ... END

обычно взаимодействует с подпрограммами обработки ошибок системы Clipper, тексты которых поставляются в файлах ERROR.PRG и ALTERROR.PRG (отладочный вариант).

При вложении операторов цикла по всей таблице SCAN ... ENDSCAN (PAL) друг в друга (скажем, для каждой записи одной таблицы просматриваются все записи другой) будьте внимательны с установкой текущей таблицы. Если сразу за заголовком оператора SCAN следует оператор цикла FOR, вставьте условие: SCAN FOR TRUE.

5.2. Подпрограммы

Способы описания и вызова подпрограмм (табл. 15) несколько отличаются в рассматриваемых системах.

Таблица 15. Использование подпрограмм

Назначение	Paradox	Clipper
Подпрограмма	PROC [CLOSED] <имя> (<список формальных параметров>) [USEVARS <список общих переменных>] [PRIVATE <список локальных переменных>] <операторы> [RETURN [<выражение>]] ENDPROC	(PROCEDURE |FUNCTION) <имя> [PARAMETERS <список формальных параметров>] [PRIVATE <список локальных переменных>] [PUBLIC <список общих переменных>] <операторы> ([RETURN]| RETURN <выражение>)
Вызов функции	<имя> (<список фактических параметров>)	<имя> (<список фактических параметров>),
Вызов процедуры	<имя> (<список фактических параметров>)	DO <имя> WITH <фактические параметры>
Вызов процедуры без аргументов	EXECPROC <выражение>	DO <имя>
Число переданных параметров	отсутствует	pCount ()
Выход из приложения	QUIT [<сообщение>]	QUIT или CANCEL

Если отвлечься от синтаксиса, работа с подпрограммами в языке Clipper очень похожа на то, к чему мы привыкли в других языках программирования. Остановимся на отличиях. Предложением Public описываются переменные, доступные из любых программных единиц, в которых те же переменные определены в списке Public. Private переменные доступны не только внутри единицы описания, но и в вызываемых подпрограммах. Число переданных параметров может быть меньше, чем определено в подпрограмме, в этом случае возможен контроль функцией pCount (). Если в качестве фактического параметра подставляется выражение, то передается значение, в противном случае передается ссылка.

Как видно из таблицы, в Paradox различие между процедурой и функцией определяется только тем, возвращается ли при выходе из подпрограммы какое-либо значение. Способ передачи параметров в подпрограммы - только по значению. Различаются два типа организации процедур: обычные и закрываемые (CLOSED). Внутри обычных подпрограмм доступны все PRIVATE переменные, определенные в вызывающих подпрограммах, а также переменные без описания (чтобы сделать переменную глобальной, не требуется никаких описаний). Для закрываемых процедур доступность переменных определяется либо в списке параметров, либо через предложение USEVARS. Может показаться, что закрываемые процедуры менее удобны, но только для них Paradox организует автоматическое управление загрузкой в память и освобождение из памяти по мере надобности.

В Paradox понятие библиотеки отличается от привычного (наиболее близкая ассоциация - это динамические библиотеки Windows). Использование библиотек, по сравнению с обыкновенными программами на языке Paradox, более предпочтительно, поскольку при работе приложения не требуется времени на трансляцию, не засоряется память неиспользуемыми процедурами, а кроме того, появляется возможность не предоставлять конечному пользователю исходных текстов процедур.

Операторы для работы с библиотеками представлены в таблице 16.

Процедуры загружаются из библиотек и освобождаются из памяти либо явными командами (ReadLib, Release Pprocs), либо автоматически - через указание системной переменной

Таблица 16. Работа с библиотеками в Paradox

Команда	Назначение
CREATELIB <библиотека>	Создание библиотечного файла с именем <библиотека>
INFOLIB <библиотека>	Выдача оглавления библиотеки в таблицу LIST
READLIB <библиотека> <список процедур>	Чтение процедуры в память
RELEASE PROCS {ALL | <список процедур>}	Освобождение процедур из памяти {всех или в соответствии со списком}
WRITELIB <библиотека> <список процедур>	Запись процедур в библиотеку
AUTOLIB = <список библиотек>	Назначение библиотек для поиска и автоматической загрузки процедур

AutoLib списка библиотечных файлов для поиска процедур и организации закрываемых (closed) процедур.

5.3. Управление памятью

Под управлением памятью в данном разделе понимается контроль свободной памяти, освобождение и восстановление значений переменных (табл. 17).

Таблица 17. Управление памятью

Назначение	Paradox	Clipper
Ограничение снизу памяти под процедуры	SETSWAP <число байтов>	Отсутствует
Удаление переменных из памяти	Release Vars {All | список}	Release {список | All [Like|Except шаблон]}
Сохранение значений переменных в файле	SaveVars {All | список}	Save to файл { All | [Like|Except шаблон]}
Чтение переменных из файла	Play "Savevars"	Restore from файл [Additive]
Объем свободной памяти	MemLeft () (в байтах)	Memory (0) (в Кбайтах)

Команда SetSwap языка PAL необходима, потому что свободная память в Paradox расходуется с двух сторон во встречных направлениях: для кода процедур и переменных - с одной, для буферов таблиц - с другой.

Clipper позволяет при записи в файл и освобождении значений переменных использовать шаблон (например, cl* - все переменные, начинающиеся с cl), а при восстановлении - очищать все остальные (команда Restore без опции Additive).

Команда сохранения SaveVars языка PAL создает файл с именем SAVEVARS.sc, содержащий операторы присваивания для сохраняемых переменных.

5.4. Отладка программ

5.4.1. Отладка программ в Paradox

Отладчик вызывается в следующих случаях: при запуске в отладчике (клавиша Alt-F10, затем - {Scripts} {Debug} и имя файла), по оператору DEBUG в теле программы, по клавише Ctrl-Break в момент выполнения программы на языке Paradox и при возникновении какой-либо ошибки (в последнем случае выдается сообщение об ошибке и меню из двух вариантов: {Cancel} и {Debug}).

Для отладки библиотечных процедур необходимо присутствие файла исходного текста программы на языке Paradox. Работающий отладчик высвечивает в предпоследней строке экрана имя программы на языке Paradox, номер текущей строки и подсказку для выхода (Ctrl-Q), в последней строке экрана - текущую строку программы; текущий оператор выделяется треугольничком слева от него. При этом можно работать в Paradox или вызывать команды отладчика. Например, если ошибка - неверно нажатая клавиша меню, можно нажать требуемую, затем проигнорировать текущий оператор программы и продолжить выполнение или предварительно просмотреть таблицы, подкорректировать данные и т.п..

Команды отладчика могут быть вызваны через меню отладчика (клавиша Alt-F10, затем первая буква альтернативы) или клавишами "быстрого вызова" (клавиша Ctrl и первая буква). Меню отладчика высвечивается в первых двух строках экрана (как и все меню Paradox версий ниже 4.0) и содержит следующие варианты: Value (ввести и вычислить значение выражения, результат - в окне сообщений), Step (выполнить очередной оператор), Next (пропустить очередной оператор), Go (продолжить выполнение без отладчика), MiniScript (ввести и выполнить строку операторов), Where? (выдать вложенность программ и процедур вместе со значениями переменных), Quit (закончить отладку, выйти в Paradox), Pop (вернуться в предыдущий контекст), Editor (остановить отладку и перейти в редактирование текущей программы). Для варианта MiniScript клавиши "быстрого вызова" не предусмотрено.

При выходе из отладчика в редактор (Ctrl-E) курсор устанавливается на строку, которая является текущей в момент вызова. А поскольку в редакторе есть команда Go (закончить редактирование и выполнить программу), то автоматически организуется цикл редактор - отладчик - редактор, что очень удобно.

5.4.2. Отладка программ в Clipper

Для включения отладчика в код программы следует в задание на компоновку добавить файл DEBUG.OBJ (поставляется с системой), а для обращения к нему надо либо включать в текст программы вызовы функции ALTD (), либо в момент работы приложения нажимать комбинацию клавиш Alt-D. При этом появляется всплывающее (pop-up) меню отладчика, содержащее разделы Control (управление), Display (высветить), Variable (переменные), Help (помощь), Break (список точек останова), Watch (показать значение) и окошко с именем текущей процедуры, номером строки исходного текста и статусом точки останова (on/off - включено/выключено).

Для обращения к большинству разделов и пунктов меню существуют клавиши "быстрого вызова" - комбинация клавиши Alt и буквы: для разделов, соответственно, Z, X, V, H, B, W, а для отдельных пунктов - S (выполнить одну строку программы и вернуться в отладчик), G (продолжить выполнение программы), P (работа со списком Watch элементов), Q (закончить программу) и некоторые другие.

При вызове watch списка надо выбрать альтернативу Add или Delete для добавления или удаления элемента из списка. При удалении элемента указывается номер, при добавлении вводится выражение, подлежащее вычислению и выводу в соответствующее окно. Удаление элемента списка точек останова (break point) осуществляется аналогично, а добавление - в одном из двух вариантов: Line и Expression. В первом случае вводится имя подпрограммы и номер строки исходного текста для останова, когда управление попадает на указанную строку (строка 0 - начало подпрограммы). Во втором вводится логическое выражение, при истинности которого управление передается в меню отладчика.

В разделе Display следующие пункты: Expression (вычислить значение выражения), Trace (вложенные вызовы подпрограмм), Status (значения системных установок), Database (информация об открытых базах данных: номера областей, связи, индексы, фильтры, структуры и прочее).

Раздел отладчика Variable позволяет просмотреть и изменить значения всех доступных переменных.

В версии Clipper 5 отладчик позволяет увидеть также исходный текст программы, а вызов программы для отладки осуществляется отдельной утилитой.

В целом отладчик системы Clipper довольно удобен и имеет ряд преимуществ перед отладчиком Paradox, в частности, watch списки, точки останова и остановы по условию; не хватает возможностей просмотреть таблицы, проигнорировать ошибочную команду или выполнить команду языка, как это допустимо в Paradox.

5.5. Примеры программ

По исходным текстам программного кода сложно оценить эффективность приложений в разных СУБД, поскольку "за кадром" остаются прочие компоненты и "внешний вид" работающих приложений. Приведенный ниже пример позволит, тем не менее, ознакомиться со стилем программирования для рассматриваемых систем.

При работе с базами данных довольно часто возникает задача поддержки таблиц-словарей, содержащих допустимые значения каких-либо полей других таблиц (назовем их основными) и расшифровки значений этих полей, например, кода города и наименования города. Предположим, что пользователь, работая с основной таблицей, видит также развернутые значения из

словаря, а при попытке ввести значение попадает в словарь для выбора подходящего значения.

Рассмотрим поддержку реализации словаря в обеих системах. Общие требования: процедура (назовем ее Dictionary) должна иметь на входе имя основной таблицы, имя текущего поля и значение текущего поля. Процедура должна найти во вспомогательной таблице CONNECT соответствующий словарь и поле связи, предоставить словарь для просмотра и корректировки, установив предварительно текущую запись в соответствии со значением поля основной таблицы, а при нажатии пользователем клавиши выбора (пусть будет F2) подставить значение поля связи из текущей записи в основную таблицу. По клавише ESC должен быть выход из словаря без подстановки значения.

Исходные тексты процедур приведены в предположении, что к моменту вызова процедуры Dictionary все таблицы открыты, основная таблица связана с каждым словарем через одно поле связи, по которому словари проиндексированы, совпадающее по типу и длине с полем основной таблицы.

Для программы на языке PAL (листинг 1) делается предположение, что для словарей подготовлены и установлены экранные формы или (как минимум) имена полей на русском языке.

В программе на языке Clipper (листинг 2) таблица корректируется функцией dbEdit, одним из параметров которой является массив заголовков для столбцов. В данном варианте реализации заголовки (массив RusFields) считываются из вспомогательной таблицы RusNames, как и сами имена полей (массив LatFields), в соответствии с порядком расположения в таблице.

Программы, естественно, несколько упрощены в целях наибольшей ясности текста.

5.6. Рекомендации

5.6.1. Clipper

Имея в виду ряд неудобств при разработке приложений в системе Clipper (см. раздел 1.2), можно предложить несколько советов. Первое. Использовать при работе два или три компьютера (лучше в составе сети) для раздельной работы редактора текстов и компиляции /компоновки/ выполнения программы. Второе. Неоценим резидентный гипертекстовый справочник по системе (Norton Guide). Третье. Связывание быстрее всего осуществляют компоновщики фирмы Microsoft: LINK.EXE из состава DOS работает примерно в два раза быстрее, чем PLINK86 из пакета Clipper, а LINK из C версии 5.0 и выше или QUICK C - почти в четыре раза. RTLINK.EXE из Clipper 5.0 работает еще более медленно, но имеет неоспоримое преимущество - связывание с ключом /D позволяет модулям любых размеров работать в очень малом объеме памяти (менее 350 Кбайтов) за счет реализации механизма подкачки объектов. Четвертое. При разработке больших систем трудно обойтись без утилиты поддержки проекта типа RMAKE.EXE из Clipper 5 или подобных. Для подготовки небольших приложений можно порекомендовать пакетный файл, содержащий вызовы редактора, компилятора и так далее, с анализом кода завершения и передачей управления на нужную метку.

Новичкам системы Clipper хотелось бы указать на несколько существенных моментов.

Во-первых, строки символов допускают концевые пробелы, т.е. "A" и "A " - не одно и то же.

Во-вторых, значение индексного выражения для поиска или установки связи с зависимой таблицей должно соответствовать по длине и промежуточным пробелам индексному выражению в момент формирования индексного файла. Например, если индексное выражение T+F использует текстовые поля длиной соответственно 4 и 5 символов, то для поиска записи со значениями этих полей "A" и "B" требуется команда

```
Seek <выражение>,
```

где <выражение> равно "A...B...." (для наглядности пробелы заменены на точки).

В-третьих, поскольку числовые поля в системе могут быть представлены с различным числом значащих цифр до и после запятой (например, N2.0 или N8.4), а числовые переменные памяти представляются 10 позициями до точки и не более чем 10 позициями после точки (с учетом общего числа значащих цифр не более 16), то при использовании числовых переменных в функциях преобразования типа возможны сюрпризы. Пример: если выражение transform(COST,"") содержит поле COST типа N3.1, то результат - строка символов длиной 3. Если же COST - это переменная памяти со значением, предположим, 1.01, то длина результирующей строки будет уже 13 символов! Когда подобное выражение - часть формулы индекса, то ошибку можно вылавливать долго.

И последнее. Если программа на языке Clipper зациклилась, для экстренного выхода существует клавиша Alt-C, после чего в первой строке экрана высветится имя текущей процедуры и вопрос - "Continue ?". После нажатия клавишы "N" программа выходит в DOS.

5.6.2. Paradox

Пользователям Paradox трудно дать существенные рекомендации. Тем не менее несколько мелких советов приведу. Во-первых, для больших приложений удобнее работать с закрываемыми процедурами. Во-вторых, что касается процедур с существенным объемом данных для инициализации массивов или текстовых строк, выносить эти операторы в отдельные процедуры (естественно, закрываемые). В-третьих, заранее резервировать память под открываемые таблицы оператором SetSwap из расчета по 2 Кбайта на открытый образ задачи. В-четвертых, иметь в виду, что процедура не может быть выгружена, если ее вызов стоит на месте фактического параметра другой процедуры. Ну и последнее - для оптимальной работы желателен защищенный режим процессора и память от 2 Мбайтов, а также не помешает кэширование диска утилитой SMARTDRV.exe или какой-либо подобной.

6. Заключение

Конечно, рамки статьи не позволяют рассмотреть все интересные моменты. "За кадром" остались такие интересные компоненты, как графы, кросс-табуляция, формирование отчетов в Paradox, и огромное число функций и команд для работы с массивами, файлами, аппарат препроцессора, блоки кода и многое другое в Clipper.

Тем не менее, читатель вправе, наконец, спросить: "Ну и где же вывод, какая СУБД лучше - Clipper или Paradox ?" А нет такого вывода (по крайней мере, у автора статьи). Лично мне более симпатичен Paradox (в целом), однако при необходимости разработать очередное приложение я не знаю, какую СУБД предпочту. Надеюсь, что эта статья поможет кому-то из читателей сделать выбор. ■

С целью предоставления владельцам персональных компьютеров информации о фирмах, действующих на компьютерном рынке, и предлагаемых ими продуктах и услугах "Монитор" публикует на страницах журнала каталоги фирм и программ.

В каталоге фирм помещается следующая информация о фирме.

- Название фирмы, адрес, телефон, факс.
- Краткое описание продуктов и услуг, предлагаемых фирмой (не более 30 слов).

В каталоге программ помещается следующая информация о программе.

- Наименование программы.
- Как связаться с поставщиком (адрес, телефон, факс).
- Краткое описание программы и ее функций (не более 30 слов).
- Аппаратура и операционная система, необходимые для работы программы.
- Языки, на которых доступны программа и документация.
- Число пользователей в России и в других странах.
- Цена (или диапазон цен).

"Монитор" публикует также объявления фирм, желающих найти высококвалифицированных специалистов по компьютерам и программированию, и объявления частных лиц, ищущих работу в этой области.

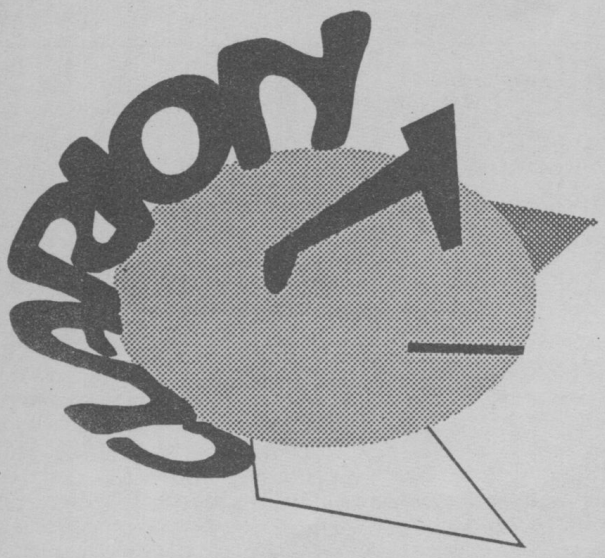
Для получения более подробной информации Вы можете позвонить по телефону (095)238-99-33 либо выслать открытку со своими координатами по адресу 117419, Москва, а/я 765.

Листинг 1

```
proc CLOSED Dictionary (OldTable, OldField, Val) ; = = = Работа со словарем
usevars AutoLib          ; — для вызова процедур
private RetField, RetTable, TmpVal

    if SysMode () < > "Edit" then ; — должен быть режим корректировки
        CoEditKey
    endif
    Moveto [Connect -> TableM] ; — поиск таблицы-словаря и поля значения
    locate oldTable, oldField
    if not retval then
        Beep
        message "Не установлена связь для данного поля"
        sleep 1500
        moveto oldTable
        return
    endif
    RetTable = [TableCon]          ; таблица - словарь
    RetField = [FieldCon]          ; поле связи
    moveto RetTable
    locate Val                     ; — переход к записи со старым значением
    While True
        Wait Record
            Prompt    " F2 - выбор текущей записи, Esc - выход без выбора",
                " Home / End, PgUp / PgDn, Ins / Del, F6: поиск "
            Until     "F2", "Esc", "Ins", "Del", "Down", "Up", "Home", "End",
                "PgUp", "PgDn", "F6"
        ; — идентификатор клавиши выхода - в переменной RetVal
        Switch
            case retval = "Esc" or retval = "F2" : QuitLoop
            case retval = "Del" :
                if getyesno ("Удалить текущую запись") then
                    Del
                endif
            case retval = "F6" :; — поиск значения
                @ 0, 0 clear Eol
                @ 1, 0 clear Eol
                @ 0, 30 "Поиск будет вестись со следующей записи."
                @ 1, 35 ?? "Введите значение для поиска -> "
                accept FieldType() required to tmpval
                Down
                locate next pattern tmpval
                if not retval then
                    Style Attribute SysColor(18)
                    @ 1, 0 clear Eol
                    @ 1, 48 ?? " Поиск закончился неудачно ..."
                    Style Attribute SysColor (9)
                    sleep 2000
                endif
        otherwise:      ; — остальные клавиши выполнит Paradox
            execute retval
        EndSwitch
    EndWhile
    if retval = "F2" then
        moveto field RetField
        Val = []
        moveto oldTable
        [] = Val
        PgDn PgUp
    else
        moveto OldTable
    endif
endproc
WRITELIB LibName Dictionary
;-----------------------------------------
proc CLOSED GetYesNo (mes); = = = Вернуть ответ Да/Нет
    private yes, no, choice
    yes = " Да "
    no = " Нет "
    ShowMenu
        yes : mes + " ?" + yes,
        no : mes + " ?" + no
        default no
        to choice
    return (choice = yes)
endproc
WRITELIB LibName GetYesNo
```

Листинг 2

```
procedure Dictionary        && = = = = Работа со словарем
    parameters OldTable, OldField, OldVal
    private RetTable, RetField, n_fields, LatFields, RusFields, RetRecord, i
    select Connect              && — поиск таблицы-словаря и поля значения
    seek oldTable + oldField
    if .not. found ()
        PutMes with "Не установлена связь для данного поля"
        select &OldTable
        return
    endif
    RetTable = TableCon             && таблица - словарь
RetField = FieldCon                 && поле связи
    select RusNames     && — вспомогательная таблица имен полей
    seek RetTable
    n_fields = 0    && подсчет числа полей (если все, то n_fields = aFields())
    do while Table_Name = RetTable
        n_fields = n_fields + 1
        skip
    enddo
    declare LatFields [n_fields], RusFields [n_fields]
    seek RetTable
    for i = 1 to n_fields
        LatFields [i] = Lat
        RusFields [i] = Rus
        skip
    enddo
    select &RetTable
    seek Val
    if .not. found ()               && значение не содержится в словаре
        goto top
    endif
    RetRecord = 0
    set color to W+/B
    clear
    @ 22, 1 say     " F2: выбор текущей записи, Esc: выход без выбора" +;
                    " Enter: корректировка"
    @ 23, 1 say " Home / End, PgUp / PgDn, Ins / Del, F6: поиск "
    set color to W+/G,W+/R,,,I/B
    @ 3, 0, 24, 79  box ("         ")          && рамка вокруг таблички
    set cursor off          && текущее поле будет выделяться только цветом
    dbedit (4, 1, 23, 78, LatFields, "dbEditKey",;
                "", RusFields, "-", "|", " = ")

    select &OldTable
    if RetRecord > 0
        replace &OldField with &RetTable -> &RetField
        skip -1
        skip
    endif
        && ————  end of Dictionary  ————
function dbEditKey          && = = = = обработка клавиш для dbEdit = = = =
    parameters mode, fld_n
    private key, retval, fld, val
    retval = 1                  && по умолчанию: продолжить обработку
    key = lastkey ()        && код последней нажатой клавиши
    do case
        case mode = 1       && попытка выхода за начало файла
            PutMes (" Начало таблицы ")
        case mode = 2       && попытка выхода за конец файла

            if GetYesNo ("Конец файла. Добавить запись ?")
                append blank
                retval = 2 && продолжение обработки, перерисовать экран
            endif
        case mode = 3       && —-  пустой файл
            if GetYesNo (" Таблица пустая. Вставить запись ?", 1, 28)
                append blank
                retval = 2
            else
                store 0 to retval, nRecordRet  && выход
            endif
        case key = 27       && — клавиша Esc : выход
            store 0 to retval, nRecordRet       && выход
        case key = 22       && — клавиша Ins : вставить запись
            if GetYesNo("Вставить запись ?")
                append blank
                retval = 2
            endif
        case key = 7        && — клавиша Del : удалить запись
            if GetYesNo ("Удалить запись ?")
                delete
                skip -1
                if bof ()
                    goto top
                else
                    skip
                endif
                retval = 2
            endif
    && клавиша Enter или буквенно-цифровая : корректировка поля
        case key = 13 .or. (key > 31 .and. key < 255)
            fld = LatFields [fld_n]
            tmpval = &fld       && переменная надежнее, чем поле
            set cursor on       && на время корректировки
            @ row(), col() get  tmpval
            if key < > 13
                keyboard chr(key)   && поместить код клавиши в буфер
            endif
            read
            replace &fld with tmpval
            set cursor off
            keyboard chr (4)            && перейти вправо
        case key = -1           && F2 :  выбор записи
            RetRecord = recno ()
            retval = 0
        endcase         && все остальные клавиши обработает dbEdit ()
    return retval
        && ————  end of  dbEditKey  ————
function  GetYesNo          && = = = = = возвращение ответа да/нет
    parameters ss
    set message to 0 center         && строка 0 - для подсказки
    @ 1, 1 prompt yes message ss + yes
    @ 1, 8 prompt no  message ss + no
    menu to choice      && возвращение номера альтернативы или 0 если Esc
    return (choice = 1)
        && ————  end of  GetYesNo  ————
procedure PutMes                && = = = = = выдача сообщения
    parameters ss
    @ 1, 0
    @ 1, 1 say ss
    inkey (5)           && ожидание истечения 5 секунд или нажатия клавиши
    @ 1, 0
        && ————  end of  PutMes  ————
```



Модельный файл Clarion

Л. Ливеровский

Прежде чем приступить к рассмотрению назначения и устройства модельного файла интегрированной инструментальной системы программирования Clarion, вкратце остановимся на обычной последовательности действий при создании приложения в среде Clarion.

Приложения в среде Clarion создаются, как правило, следующим образом.

С помощью программы Дизайнер (Designer) разрабатывается общая структура программы (дерево вызовов) и проектируются модули интерфейса с человеком-оператором и формы отчетов.

В режиме диалога без программирования Дизайнер позволяет создать три типа процедур интерфейса с человеком-оператором и процедуру-отчет:

- меню (Menu);
- таблица (Table) - просмотр файла в виде таблицы и вызов процедур удаления, добавления и измения; ширина таблицы не может быть больше ширины экрана и горизонтальный скролинг не осуществляется;
- экран (Form) - экранная форма для редактирования полей и переменных с возможностью ввода значения в поле из другого файла, обычно называемого справочником (look up fields);
- отчет (Report).

Более сложные действия по содержательной обработке данных программируются вручную, и процедуры, осуществляющие эти действия, включаются в общую структуру приложения с типом OTHER (другой).

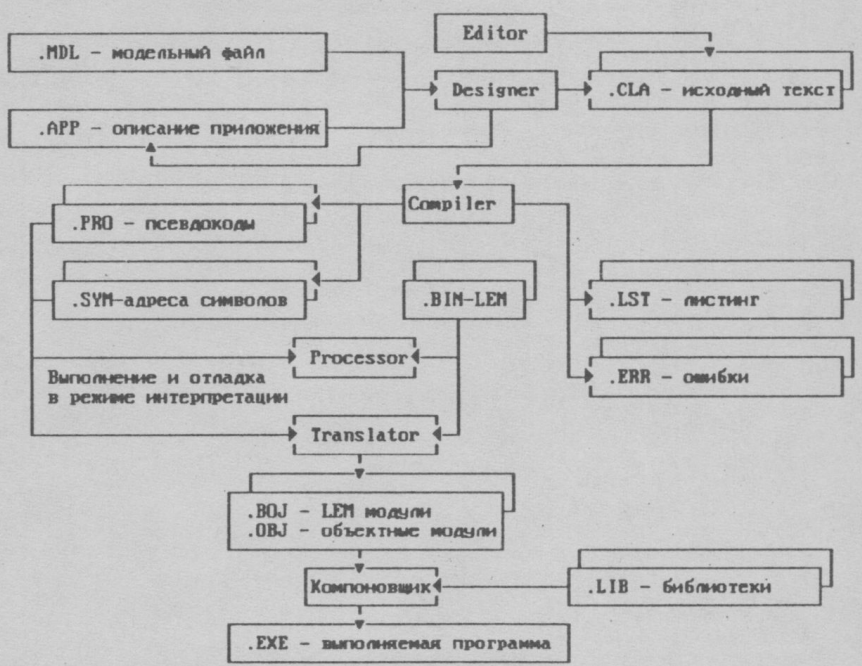


Рис. 1. Общая схема создания приложения в среде Clarion

Вся информация о приложении содержится в файле описания приложения, который имеет расширение APP. По окончании проектирования приложения Дизайнер, используя так называемый модельный файл (имеющий расширение MDL), генерирует исходные тексты программ.

Эти файлы транслируются компилятором (Compiler) и могут выполнятся в режиме интерпретации под управлением программы Processor.

Выполняемый EXE модуль также может быть создан после обработки транслятором (Translator), который создает файлы управления компоновкой для программ LINK, PLINK86, RTLink, по выбору пользователя, и осуществляет собственную сборку.

Общая схема последовательности разработки приложения в среде Clarion показана на рис. 1.

Модельный файл содержит макроопределения, по которым строится процедура каждого типа (меню, таблица, экран и отчет). Стандартно в составе пакета поставляются два модельных файла: для обычных (STANDARD.MDL) и работающих в сети приложений (NETWORK.MDL).

Структура и назначение модельного файла

Основное назначение модельного файла, по мысли разработчиков Clarion, сосредоточить описания способов порождения каждого типа генерируемой Дизайнером процедуры в модельном файле и сделать программу Дизайнер независимой от конкретного кода, требующегося для реализации процедур, им поддерживаемых. Конкретные параметры, необходимые для генерации исходного текста, находятся при этом в файле описания приложения (это описание в режиме диалога с программистом создает Дизайнер). В идеале для достижения этой цели макроопределения модельного файла в процессе макрорасширения (подстановки в макроопределения значений настраиваемых параметров генерируемых процедур) должны полностью контролировать создание исходного текста процедур. В этом случае Дизайнер, проводя генерацию исходных текстов, должен был бы работать как полнофункциональный макрогенератор.

Однако разработчикам системы Clarion полностью достичь своей цели пока не удалось. Некоторые решения по генерации исходного текста принимает программа Дизайнер, и эти решения не контролируются модельным файлом.

Модельный файл содержит модельные процедуры (model procedure) для каждого типа процедуры, поддерживаемой Дизайнером. Они используются при генерации исходных текстов всех процедур соответствующего типа по умолчанию. Возможно вместо стандартной модельной процедуры для каждой конкретной процедуры указать собственное имя модельной

процедуры. В этом случае такую модельную процедуру должен подготовить пользователь и включить ее в модельный файл.

Модельная процедура представляет собой комбинацию программного псевдокода и реального кода.

Псевдокод содержит распознаваемые Дизайнером ключевые слова (keywords), которые замещаются соответствующим исходным текстом, базирующимся на параметрах генерируемой процедуры из файла описания приложения. Это замещение проводится по правилам простой подстановки. В самом модельном файле нет условных и других сложных макрооператоров, характерных для развитых макроязыков. Надо отметить, что ключевые слова не обязательно заменяются каким-либо кодом. Это зависит от описания конкретной процедуры в файле описания приложения. Решения о замещении принимает программа Дизайнер, основываясь на своих внутренних правилах. (Если ключевое слово не заменяется на что-нибудь, то Дизайнер может не включить всю строку модельной процедуры, содержащую это ключевое слово, в результирующий исходный текст.)

Реальный код включается в генерируемый исходный текст без изменений.

Источником информации для замещения ключевого слова может являться не только файл описания приложения, а и так называемая ключевая процедура (keyword procedure), расположенная в модельном файле. Это позволяет, во-первых, хранить необязательный код отдельно от модельной процедуры, во-вторых, несколько раз включать текст, определяемый ключевым словом, в модельную процедуру, в-третьих, гарантировать, что определенные операции производятся единообразно во всем приложении.

Разработчики системы Clarion обещают, что в последующих версиях системы модельный файл будет содержать все больше возможностей по управлению генерацией исходного текста, а Дизайнер будет принимать все меньше решений относительно порождения кода.

Синтаксис модельного файла

Модельный файл состоит из модельных и ключевых процедур. Модельная или ключевая процедура:

```
*<имя процедуры>
...
<реальный код и псевдокод>
...
*<имя следующей процедуры>
...
<реальный код и псевдокод>
...
```

Имя процедуры отмечается звездочкой (*) в первой позиции. Имя может быть длиной до 12 символов и содержать буквы и цифры. Начинается имя всегда с буквы.

Дизайнер включает в исходный текст строки из модельного файла от начала модельной или ключевой процедуры до первой звездочки в первой колонке или до конца файла.

В процессе включения строк в исходный текст Дизайнер производит подстановки кода вместо ключевых слов. Дизайнер сам определяет, какие ключевые слова чем замещаются, и пропускает необязательные, если для этой процедуры они не нужны. Все ключевые слова начинаются с символа @.

Модельный файл содержит четыре модельные процедуры (model procedure) для каждого типа процедуры, поддерживаемой Дизайнером (Menu, Form, Table, Report), и две дополнительные:

- MEMFORM включается, если процедура типа FORM работает с таблицей в памяти, а не с файлом;
- MEMREPORT включается, если процедура типа REPORT работает с таблицей в памяти, а не с файлом.

Простейшая модельная процедура выглядит так:

```
*SIMPLEST
@PROCNAME PROCEDURE
CODE
*
```

SIMPLEST - имя модельной процедуры.

@PROCNAME - ключевое слово, соответствующее имени процедуры. Замещается на имя, заданное параметром Procedure Name в окне опций процедуры (Options Window) Дизайнера.

Если в параметре модельной процедуры (Model Procedure) в окне опций процедуры (Options Window) Дизайнера задать имя SIMPLEST и включить эту модельную процедуру в модельный файл, а в параметре имени процедуры (Procedure Name) задать имя EXAMPLE, то после генерации исходного текста получится следующий код:

```
EXAMPLE PROCEDURE
CODE
```

Всего Дизайнер распознает около 110 ключевых слов модельного файла. Все они соответствуют определенным параметрам генерируемой процедуры. Например, ключевое слово @SCREENOPT замещается на описания атрибутов окна: WINDOW(<ряд>,<кол>), AT(<ряд>,<кол>), HUE(<цвет>,<цвет фона>), HLP(<идентификатор>).

Также Дизайнер распознает около 40 ключевых процедур, управляющих генерацией разнообразных частей кода, стандартных и часто повторяющихся в процедурах разного типа.

В связи с ограниченностью объема журнальной статьи описание ключевых слов и ключевых процедур здесь не приводится. Подробно с назначением ключевых слов и ключевых процедур можно познакомиться, прочитав содержимое файла MODEL.DOC и посмотрев сам модельный файл STANDARD.MDL или NETWORK.MDL.

Изменение модельного файла

При проектировании приложения часты случаи, когда в целях получения необходимого кода удобнее или просто нужно изменить какую-то модельную процедуру.

Например, если пользователя не устраивают какие-то фрагменты кода, создаваемые Дизайнером, то можно необходимые изменения внести в текст сгенерированной процедуры вручную. При этом повторная генерация исходных текстов уничтожит внесенные ранее изменения. Если пользователю нужно несколько раз проходить этап проектирования программы с помощью Дизайнера для внесения уточнений и исправлений (а это всегда так) и/или эти изменения необходимо вносить в целый ряд однотипных процедур (что тоже бывает очень часто), то для каждой процедуры, в которую вносятся изменения, вероятно, лучше создать собственную модельную процедуру.

Делается это следующим образом. В модельном файле выбирается наиболее подходящая стандартная модельная процедура (например, для экранной формы это будет процедура FORM). Она дублируется, ей присваивается новое имя (например, NEWFORM), и в нее же вносятся необходимые поправки.

Следует помнить, что пользователь не может создавать новые ключевые процедуры, так как их имена и назначение "зашиты" в программу Дизайнер. Пользователь может изменить содержимое ключевой процедуры. Но делать это нужно с крайней осмотрительностью, так как они используются при генерации текстов процедур разного типа и могут использоваться неявно. Пользователь, естественно, не может определять и свои ключевые слова.

Рассмотрим простой пример, в котором покажем, почему необходимо создать свою модельную процедуру и как это сделать. Описанный вариант является не единственным и, возможно, не оптимальным. Это только иллюстрация принципа работы с модельным файлом.

Пусть имеются три файла, структуры которых показаны на рис. 2, 3 и 4. Для каждой записи в главном файле сотрудников PERSON в файле начислений SALARY может быть 0 или более записей о начислениях заработной платы сотруднику. Структура демонстрационной программы дана на рис. 5.

Операции добавления, редактирования записей в файле PERSON реализуются средствами Дизайнера с использованием стандартного модельного файла.

Операция удаления записи в файле PERSON требует удаления 0 или более записей в файле SALARY. (Удаляются все записи со значением поля TAB_NO, равным значению поля

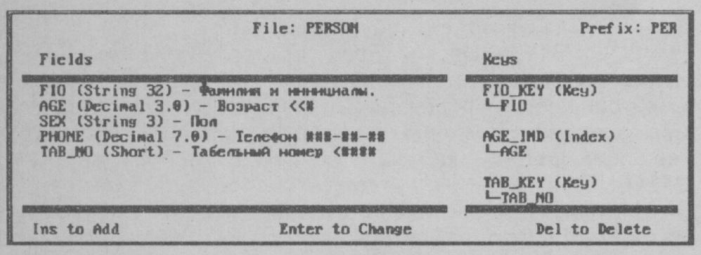


Рис. 2. Файл сотрудников

TAB_NO удаляемой записи файла PERSON.) С помощью Дизайнера такую процедуру создать нельзя. Поэтому ее необходимо закодировать вручную. (Процедура удаления всех записей о заработках DEL_SALARY помечена типом Other в дереве вызовов.)

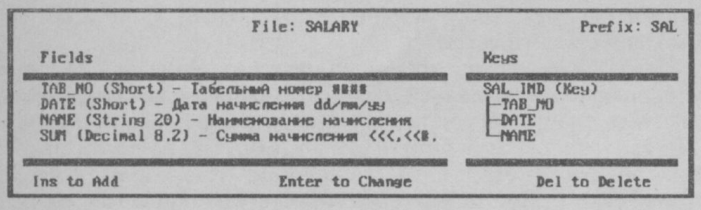


Рис. 3. Файл начислений заработной платы сотрудникам

Теперь возникает вопрос, как сделать так, чтобы Дизайнер поместил вызов процедуры DEL_SALARY до удаления записи файла PERSON и произвел проверку результатов ее работы, чтобы, во-первых, был известен табельный номер удаляемого сотрудника и, во-вторых, чтобы в случае ошибки в процедуре DEL_SALARY не удалить запись из файла PERSON. Когда удалена запись из основного файла PERSON, а из файла SALARY соответствующие записи не удалены, появляются "висящие ссылки" по табельному номеру, что весьма неприятно.

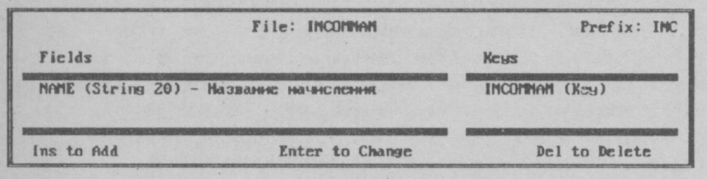


Рис. 4. Справочник начислений

Для процедуры типа FORM воспользуемся параметром следующей процедуры (Next Procedure), который обычно используется для вызова следующей экранной формы, если форма не помещается на одном экране. Этому параметру соответствует ключевое слово @NEXTPAGE. Следующая форма вызывается с использованием ключевой процедуры NEXTFORM, где и содержится ключевое слово @NEXTPAGE.

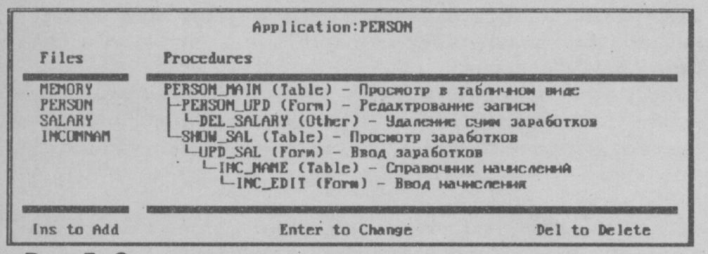


Рис. 5. Структура программы

В качестве прототипа нашей модельной процедуры, которую мы назовем NEWFORM, возьмем стандартную модельную процедуру FORM. Ключевое слово @NEXTPAGE будет использоваться для вызова процедуры нестандартных действий при удалении записи из файла.

Выдержки из стандартной модельной процедуры FORM, необходимые для внесения наших поправок, приведены в листинге 1.

После внесения всех изменений новая модельная процедура NEWFORM не использует ключевую процедуру NEXTFORM (листинг 2).

Для пояснения скажем, что глобальная переменная ACTION служит для указания действия, которое необходимо совершить с записью, и в нее же процедуры помещают код нормального завершения.

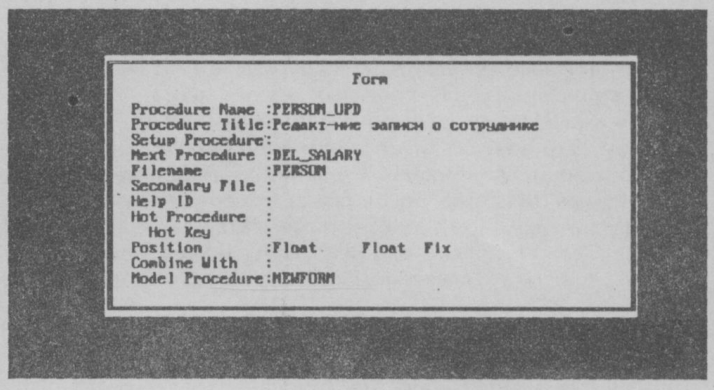


Рис. 6. Окно опций дизайна для процедуры PERSON_UPD

Перед вызовом процедуры в переменную ACTION помещаются следующие значения:

1 - добавить;
2 - редактировать;
3 - удалить.

Процедура для указания своего нормального завершения помещает в переменную ACTION значение 0.

Теперь в параметре модельной процедуры (Model Procedure) в окне опций процедуры (Options Window) Дизайнера для процедуры PERSON_UPD надо задать имя NEWFORM, и Дизайнер сгенерирует необходимый нам код (рис. 6).

Для проверки того, как Вы поняли устройство модельного файла, попробуйте ответить на вопрос: " Что будет, если для процедуры типа FORM, генерируемой с помощью модельной процедуры NEWFORM, не указать параметр следующей процедуры (Next Procedure) в окне опций процедуры (Options Window) Дизайнера ?"

Правильный ответ: исходный текст будет генерироваться, программа будет транслироваться без ошибок, но работать она будет не так, как можно было бы ожидать. ■

▼ Листинг 1. Выдержки из стандартной модельной процедуры FORM, необходимые для внесения наших поправок

```
*FORM*****************************************************
@PROCNAME PROCEDURE

<пропущено>

  LOOP                  !Цикл по всем полям ввода/редактирования

  <пропущено>

     CASE FIELD()!Обработка ввода по полям

     <пропущено>

     ! Часть кода, куда вносятся изменения
     OF ?LAST_FIELD          ! Последнее поле ввода
     EXECUTE ACTION          ! Обновить файл:
     ADD(@FILENAME)          ! добавить запись
     PUT(@FILENAME)          ! изменить запись
     DELETE(@FILENAME)       ! удалить запись

     IF ERRORCODE() = 40     ! Ошибка "двойной ключ"
     MEM:MESSAGE = ERROR()   Вывести сообщение
     SELECT(2)               ! Перейти на начало формы
     CYCLE                   ! Выйти из цикла редактирования полей
     ELSIF ERROR()           ! Проверить на другие ошибки
```

```
      STOP(ERROR())              ! Остановить программу

      PUT(@FILENAME2)            ! Обновить вторичные файлы
      PUT(@FILENAME3)            ! Обновить вторичные файлы
      PUT(@FILENAME4)            ! Обновить вторичные файлы
      @NEXTFORM                  ! Вызвать следующую форму
      ACTION = 0          ! Установить код "СДЕЛАНО"
      RETURN              ! Вернуться в вызывающую программу
  !Конец части кода, куда вносятся изменения
   <пропущено>

.
<пропущено>

Следующая форма вызывается с использованием ключевой процедуры
NEXTFORM:

*NEXTFORM*********************************************
  IF ACTION < > 3                ! Если это не удаление,
      ACTION = 2                 ! установить действие (изменить)
      @NEXTPAGE                  ! Вызвать следующую процедуру
      IF ACTION                  ! Если запись не изменялась,
          SELECT(?LAST_FIELD - 1)  ! выбрать последнее поле ввода
          CYCLE                    ! и цикл
..
```

Листинг 2. Новая модельная процедура

```
*NEWFORM*********************************************
@PROCNAME   PROCEDURE

<так же, как и в FORM>

  LOOP                ! Цикл по всем полям ввода/редактирования

  <так же, как и в FORM>

      CASE FIELD()               ! Обработка ввода по полям

      <так же, как и в FORM>

      ! Новый код
      OF ?LAST_FIELD             ! Последнее поле ввода
          IF ACTION = 3          ! Если это удаление,
              ACTOLD# = ACTION     ! сохранить значение ACTION
              @NEXTPAGE          ! Вызвать следующую процедуру
              IF ACTION          ! Если ACTION < > 0,
                  SELECT(?LAST_FIELD - 1) ! выбрать последнее поле
                  CYCLE          ! и снова цикл
              ELSE               ! Если ACTION = 0,
                  DELETE(@FILENAME)      ! удалить запись.
          .
          ACTION = ACTOLD#       ! Восстановить значение ACTION
      .


          EXECUTE ACTION         ! Обновить файл
          ADD(@FILENAME)         ! Добавить запись
          PUT(@FILENAME)         ! Изменить запись
          .
          IF ERRORCODE() = 40 ! Ошибка "двойной ключ"
          MEM:MESSAGE = ERROR()      ! Вывести сообщение
          SELECT(2)              ! Перейти на начало формы
          CYCLE                  ! Выйти из цикла редактирования полей
          ELSIF ERROR()          ! Проверить на другие ошибки
          STOP(ERROR())          ! Остановить программу
          .
          PUT(@FILENAME2)        ! Обновить вторичные файлы
          PUT(@FILENAME3)        ! Обновить вторичные файлы
          PUT(@FILENAME4)        ! Обновить вторичные файлы
          ACTION = 0             ! Установить код "СДЕЛАНО"
          RETURN                 ! Вернуться в вызывающую программу
      !Конец нового кода
      <так же, как и в FORM>


  <так же, как и в FORM>
```

Анонс

"Монитор" - версия 1.93

Д. Арапов. Пишем упаковщик

Подлинный триумф объектно-ориентированного программирования. Полный текст процедур упаковки/распаковки файлов по алгоритму LZW, реализованный на Turbo Pascal 6.0.

Л. Ливеровский. Модули расширения языка Clarion

Как включить в Clarion программу процедуры на ассемблере и Си? Особенности и ограничения межязыкового интерфейса.

В. Водолазский. Коммерческие системы шифрования: основные алгоритмы и их реализация. Часть 3

Описание алгоритмов рюкзаковой упаковки и ElGamal.

Б. Телеснин. Адаптер VGA. Режим 256 цветов

Все об "интимной жизни" VGA адаптеров с примерами на Turbo Pascal 6.0.

А. Аброськин. Снова о TSR, или как пользоваться функциями OS и BIOS из Вашей резидентной программы

Технология создания TSR программ, не конфликтующих с DOS и BIOS.

А. Петренко. Познакомимся с VDM

Рассказ о венском методе программирования.

Д. Солдатенков. Трехмерная графика

Продолжение цикла статей о программировании компьютерных игр.

Д. Пластинин. Многооконный интерфейс в среде MS Windows

Организация поддержки многооконного интерфейса в Windows 3.0. Пример его использования на Си.

Ю. Носков. Практика построения экспертных систем средствами оболочки VPX

Оболочка VPX - простое и удобное средство для разработки и отладки экспертных систем. Рассматриваются характеристики VPX и простейшие приемы использования.

С. Кириллов. Скорость исполнения и размер кода программы. Возможен ли компромисс?

Обсуждение вопросов оптимизации программ на языке Си.

И. Кравчинский. Возможности СУБД db-VISTA 3

Реализация сетевой модели данных в рамках db-VISTA 3. Примеры описания структур данных.

Несколько полезных макросов для Turbo Pascal 6.0

С.Трепыхалин

Лень - двигатель прогресса !

Есть ли что-либо более нудное, чем ввод текста новой программы, сопровождающийся многочисленными нажатиями на Alt-F9 (Compile) с последующим поиском и исправлением самых разнообразных ошибок ? И это в тот самый момент, когда не терпится побыстрее увидеть результат! Кроме того, весьма приятно (к сожалению, далеко не всем) лицезреть текст рожденной в муках программы красиво оформленным. Искусство, как известно, требует жертв, но их можно в значительной мере избежать за счет использования Turbo Editor Macro Compiler (TEMC), любезно включенного фирмой Borland в состав утилит Turbo Pascal 6.0 .

TEMC использует Turbo Editor Macro Language (TEML), дающий возможность расширить или изменить возможности редактора и даже отладчика интегрированной среды (IDE) Turbo Pascal 6.0. Используя 140 встроенных макрокоманд, Вы можете определить свои собственные, весьма изощренные клавиатурные макросы, которые позволят существенно снизить трудоемкость ввода текстов (не обязательно программ) и количество ошибок в этих текстах. В принципе, Вы даже можете реализовать многие клавиатурные комбинации более привычных Вам редакторов или отладчиков (например, Brief или CodeView).

Описание TEML и TEMC приведено в документации к Turbo Pascal и более или менее доступно пониманию среднего жителя СНГ, "читающего и переводящего со словарем". При наличии неясностей есть смысл ознакомиться с online документацией более позднего TEMC, входящего в состав Borland C++ версии 2.0 (хотя синтаксис двух TEML несколько различен).

Процесс создания собственных макросов описан предельно кратко и требует некоторых пояснений. Сначала необходимо сформировать текстовый файл макроопределений (macro script). Разумно делать это с помощью редактора Turbo Pascal . Затем macro script необходимо скомпилировать, используя TEMC.EXE. Выходным файлом TEMC будет файл конфигурации Turbo Pascal IDE (*.TP). При этом изменению подвергаются только те клавиатурные определения, которые Вы пытаетесь изменить, все прочие остаются прежними.

В реальной жизни при написании достаточно сложных макросов быстро возникает необходимость в их проверке и отладке. Поскольку отладка оных возможна только в IDE, дерзну рекомендовать следующую методику разработки и отладки макросов.

1. Убедитесь в том, что Ваш Turbo Pascal 6.0 правильно сконфигурирован (turbo.exe и temc.exe доступны через PATH и т.п.).

2. Осознайте, в какой конфигурационный файл Вы желаете внести изменения (обычно, это C:\TP\TURBO.TP), ибо, вообще говоря, каждый пользователь может иметь несколько таковых, в зависимости от решаемых задач и квалификации.

3. На всякий случай сохраните старый TURBO.TP в файле TURBO0.TP (здесь и далее все названия, кроме фирменных, условны), создайте временный каталог \MYMACROS и скопируйте в него файл TURBO.TP, переименовав в TPCONFIG.TP (для примера).

4. Скопируйте какой-либо текстовый файл в каталог MYMACROS для проведения экспериментов и сформируйте там же batch файл T.BAT (пример 1). Далее работайте в этом каталоге .

Пример 1. Отладочный файл T.BAT

```
@rem Убрать старый файл (имеет смысл при повторах)
@del tpconfig.tp
@rem Скопировать эталон в модифицируемый файл
@copy c:\tp\turbo0.tp tpconfig.tp
@rem Компилировать созданный macro script
@temc MyMacro.tem tpconfig.tp
@rem Вернуться в IDE
@exit
```

5. Запустите IDE, загрузив в первое окно вышеупомянутый текстовый файл-"жертву эксперимента", а во второе вводите свой macro script (в качестве первой "рыбы" разумно использовать пример Script из файла TEMC.DOC).

6. Отредактировав свой сценарий, запишите его на диск (для рассматриваемого примера - в файл MyMacro.TEM).

7. Нажав Alt-F-D, выйдите в DOS shell.

8. Запустите T.BAT.

9. Если не произошло никакой путаницы с каталогами, то TEMC должен скомпилировать Ваш сценарий, модифицировав УЖЕ СУЩЕСТВУЮЩИЙ в текущем каталоге файл TPCONFIG.TP , а команда EXIT должна возвратить Вас в IDE.

10.Нажмите Alt-F5, чтобы убедиться в отсутствии ошибок при компиляции сценария. Если TEMC выдал сообщение об ошибках, исправьте их и повторите этапы 6...10.

11.Нажмите Alt-O-R (Retrieve options) и выберите конфигурационный файл TPCONFIG.TP из текущего каталога (с ним-то Вы и упражняетесь). Загрузите этот файл, перейдите в первое окно и экспериментируйте.

12. Повторяйте этапы 6...11 до получения искомых результатов.

Если все сказанное не вдохновило прагматично настроенного читателя, мне остается только предложить его вниманию собственноручно изготовленный сценарий (см. листинг), реализующий заготовки наиболее часто встречающихся фрагментов программ на языке Turbo Pascal. Должен сразу же заметить, что стиль оформления программ - в значительной мере дело вкуса, однако общее правило состоит в том, что стирать лишний текст гораздо проще, чем без ошибок вводить новый.

При правильной компиляции сценария и загрузке конфигурационного файла станут доступны следующие макросы :

```
Alt-T-F  :  MakeFuncText;    /* Function */
Alt-T-P  :  MakeProcText;    /* Procedure */
Alt-T-A  :  MakeAsmEnd;      /* Asm...end;*/
Alt-T-B  :  MakeBeginEnd;    /* Begin...end;*/
Alt-T-C  :  MakeCase;        /* Case */
Alt-T-I  :  MakeIf;          /* If */
Alt-T-O  :  MakeFor;         /* fOr */
Alt-T-R  :  MakeRepeatUntil; /* Repeat...until*/
Alt-T-W  :  MakeWhile;       /* While */
Alt-T-L  :  MakeOpenFile;    /* open fiLe */
```

В результате, если при редактировании Pascal программы Вы нажмете Alt-T-B, в текст, начиная с позиции курсора, будет вставлен блок

```
begin
end;
```

При нажатии на Alt-T-C Вы увидите следующее:

```
case of
  :
else :
end; {.....of case }
```

Ну а если Вы введете в начале строки имя новой функции (например, TheBestFunctionOfMyLife) и нажмете Alt-T-F, в тексте появится последовательность:

```
function TheBestFunctionOfMyLife :  ;
{                                        }
{       TheBestFunctionOfMyLife :        }
{                                        }
begin
end; {..... of function TheBestFunctionOfMyLife }
```

В заключение следует заметить, что размеры и уровень сложности Ваших сценариев ограничены только Вашей фантазией и 64 Кбайтами. ■

Листинг

```
Script MyMacros; /* ни к чему не обязывающее название сценария */
/*                                                                */
/* Сергей [Александрович] Трепыхалин : Макросы для Turbo Pascal 6.0 IDE /*
/*                                                                */
/* Макрос MakeFuncText оформляет заголовок функции,               */
/* предполагая, что имя функции начинается в первой позиции строки*/
/* Пример :                                                       */
/* - введите имя новой функции, начиная с начала строки:          */
/* MyBestFunction                                                 */
/* - нажмите ALT+T+F                                              */
/* - на экране появится текст (комментарий в скобках введите сами): */
/*                                                                */
/* function MyBestFunction : ;                                    */
/* {-------------------------------------------------}           */
/* { MyBestFunction : (Лучшая функция всей моей жизни)  }         */
/* {-------------------------------------------------}           */
/* begin                                                          */
/* end;                                                           */
/*                                                                */
  Macro MakeFuncText
        RightOfLine;                    /* На всякий случай ... */
        InsertText(" : ;\n\n");    /* Вставить : , ; , перевод строки */
        CursorUp;CursorUp;LeftOfLine;    /* Установить курсор */
                                         /* перед именем функции */
        SetBlockBeg;                    /* Начать выделение имени функции */
        WordRight;                      /* Переместить курсор на слово вправо */
        CursorLeft;CursorLeft;CursorLeft;
        SetBlockEnd;HighLightBlock; /* Выделить имя функции */
        CursorDown;LeftOfLine;
        CopyBlock;                  /* Скопировать имя функции (дело вкуса) */
        CursorDown;LeftOfLine;
        CopyBlock;                  /* Скопировать имя функции (для end;....)*/
        LeftOfLine;InsertText("end; {..... of function ");
        RightOfLine;BackSpaceDelete;InsertText(" }\n");
        CursorUp;CursorUp;LeftOfLine;
        InsertText("{   ");         /* Сформировать элемент рамки */
        CursorUp;LeftOfLine;
        InsertText("function ");        /* Вставить "function" слева от имени */
        LeftOfLine;CursorDown;
        InsertText("{-------------------------------------------------}\n");
        CursorUp;RightOfLine;CursorDown;CursorLeft;CursorLeft; InsertText("}\n");
        InsertText("{-------------------------------------------------}\n");
        InsertText("begin");            /* Вставить "begin" */
        LeftOfLine;CursorUp;            /* Установить курсор */
        CursorUp;CursorUp;                  /* после объявления */
        CursorUp;WordRight;WordRight;   /* функции */
  end;
/* MakeProcText : оформляет заголовок процедуры аналогично MakeFuncText */
  Macro MakeProcText
        RightOfLine;                    /* На всякий случай ... */
        InsertText(" ;\n\n");           /* Вставить ; , перевод строки */
        CursorUp;CursorUp;LeftOfLine;    /* Установить курсор */
                                         /* перед именем процедуры */
        SetBlockBeg;                    /* Начать выделение имени процедуры*/
        WordRight;                      /* Переместить курсор на слово вправо */
        CursorLeft;
        SetBlockEnd;HighLightBlock;         /* Выделить имя процедуры */
        CursorDown;LeftOfLine;
        CopyBlock;                  /* Скопировать имя процедуры (дело вкуса) */
        CursorDown;LeftOfLine;
        CopyBlock;                  /* Скопировать имя процедуры (для end;....)*/
        LeftOfLine;InsertText("end; {..... of procedure ");
        RightOfLine;InsertText(" }\n");
        CursorUp;CursorUp;LeftOfLine;
        InsertText("{   ");         /* Сформировать элемент рамки */
        CursorUp;LeftOfLine;
        InsertText("procedure ");       /* Вставить "procedure" слева от имени */
        LeftOfLine;CursorDown;
        InsertText("{-------------------------------------------}\n");
        CursorUp;RightOfLine;CursorDown;CursorLeft;CursorLeft;  InsertText("}\n");
        InsertText("{-------------------------------------------}\n");
        InsertText("begin");        /* Вставить "begin" */
        LeftOfLine;CursorUp;            /* Установить курсор */
        CursorUp;CursorUp;                  /* после объявления  */
        CursorUp;WordRight;WordRight;   /* процедуры */
  end;
  Macro MakeAsmEnd              /* ASM...END; для встроенного ассемблера */
        InsertText("asm\n");
        InsertText("end;\n");
        CursorUp;CursorUp;RightOfLine;
  end;
  Macro MakeBeginEnd                    /* Это совсем просто */
        InsertText("begin\n");
        InsertText("end;\n");
        CursorUp;CursorUp;RightOfLine;
  end;
  Macro MakeIf                          /* И это тоже ... */
        InsertText("if \n");
        InsertText("  then begin\n");
        InsertText("    end\n");
        CursorLeft;CursorLeft;CursorLeft;CursorLeft;CursorLeft;
        InsertText("else begin\n");
        InsertText("    end; {.....of if  }\n");
        CursorUp;CursorUp;CursorUp;CursorUp;CursorUp;RightOfLine;
  end;
  Macro MakeCase
        InsertText("case of\n");
        InsertText("    : \n");
        InsertText("else : \n");
        InsertText("end; {.....of case  }\n");
        CursorUp;CursorUp;CursorUp;CursorUp;RightOfLine;WordLeft;
  end;
  Macro MakeFor
        InsertText("for :=  to do begin\n");
        InsertText("end; {.....of for  }\n");
        CursorUp;CursorUp;RightOfLine;WordLeft;WordLeft;WordLeft;WordLeft;
        CursorRight;CursorRight;CursorRight;CursorRight;
  end;
  Macro MakeRepeatUntil
        InsertText("repeat\n");
        InsertText("until  ;\n");
        CursorUp;RightOfLine;
  end;
  Macro MakeWhile
        InsertText("while  do begin\n");
        InsertText("end; {.....of while  }\n");
        CursorUp;CursorUp;RightOfLine;WordLeft;WordLeft;CursorLeft;
  end;
  Macro MakeOpenFile
        RightOfLine;InsertText("\n  Assign(fvar, fname);\n");
        LeftOfLine;InsertText("  {$I-}\n");
        LeftOfLine;InsertText(" Append(fvar);\n"); /* Выберите то, что нужно : */
        LeftOfLine;InsertText(" Reset(fvar);\n");/* уничтожать текст проще, */
        LeftOfLine;InsertText(" Rewrite(fvar);\n"); /* чем без ошибок вводить ! */
        LeftOfLine;InsertText("  {$I+}\n");
        LeftOfLine;InsertText(" if IOResult = 0\n");
        LeftOfLine;InsertText(" then\n");
        LeftOfLine;InsertText(" begin\n");
        LeftOfLine;InsertText("  end\n");
        LeftOfLine;InsertText(" else\n");
        LeftOfLine;InsertText(" begin\n");
        LeftOfLine;InsertText("  end;\n");
  end;
  /*---- Определение клавиш вызова макросов ----*/
  /* (приходится указывать и прописные и строчные */
  /*  буквы, так как поставляемый с TPascal 6.0  */
  /*  TEMC версии 1.0 несколько туповат .)       */
  /*  А может быть, и не он ?...                 */
  Alt-T+F       : MakeFuncText;
  Alt-T+f       : MakeFuncText;
  Alt-T+P       : MakeProcText;
  Alt-T+p       : MakeProcText;
  Alt-T+A       : MakeAsmEnd;
  Alt-T+a       : MakeAsmEnd;
  Alt-T+B       : MakeBeginEnd;
  Alt-T+b       : MakeBeginEnd;
  Alt-T+C       : MakeCase;
  Alt-T+c       : MakeCase;
  Alt-T+I       : MakeIf;
  Alt-T+i       : MakeIf;
  Alt-T+O       : MakeFor;                  /* fOr */
  Alt-T+o       : MakeFor;                  /* fOr */
  Alt-T+R       : MakeRepeatUntil;
  Alt-T+r       : MakeRepeatUntil;
  Alt-T+W       : MakeWhile;
  Alt-T+w       : MakeWhile;
  Alt-T+L       : MakeOpenFile;             /* open fiLe */
  Alt-T+l       : MakeOpenFile;             /* open fiLe */
  /*---------- конец сценария ----------*/
```

■

Имеющий уши да услышит

Д. Солдатенков

Считается, что звуковые возможности IBM PC довольно бедны и без специализированных плат добиться хорошего звукового сопровождения нельзя. Сложности в создании звука связаны с тем, что на динамик можно посылать сигнал только двух уровней - 0 или 1.

Мне кажется, для IBM PC можно выделить четыре способа создания звука.

Первый, самый простой и известный, использует для генерации сигнала микросхему таймера. При использовании этого способа можно генерировать только гудки разной частоты со скважностью 1 (скважность—это отношение времени, когда уровень сигнала равен единице, ко времени, когда уровень сигнала равен нулю), но зато музыка, составленная из этих гудков, может идти в фоновом режиме на самой слабой XT, ничуть не замедляя работу основной программы.

Можно отказаться от аппаратной генерации звука и управлять динамиком программно через 2 бит 61H порта. При этом существенно увеличиваются звуковые возможности, но возрастает и время, которое должен тратить процессор на управление динамиком.

Рассмотрим несколько способов создания звука при помощи непосредственного управления динамиком.

Первый из них интересен тем, что частота поступления данных может быть довольно низкой, приблизительно 2 КГц, и данный алгоритм хорошо работает в фоновом режиме. Идея этого способа заключается в том, что при помощи прямоугольников можно грубо воспроизвести форму огибающей звукового сигнала.

Он удобен для создания простых звуковых эффектов, но для того чтобы успешно его использовать, надо хорошо представлять себе огибающую того звукового сигнала, который хочется воспроизвести.

Посмотрим, как будет меняться характер звука, если изменять скважность сигнала, не изменяя его частоту (рис. 1).

Высота тона не меняется совсем, немного изменяется тембр и очень заметно меняется громкость, то есть возникает возможность управления амплитудой за счет изменения скважности.

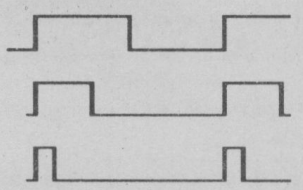


Рис. 1. Изменение скважности без изменения частоты

Огибающая звукового сигнала, производимого барабаном, имеет вид синусоиды, вписанной в затухающую экспоненту. Если мы выведем на динамик сигнал следующей формы (рис. 2), то получится звук, похожий на удар барабана.

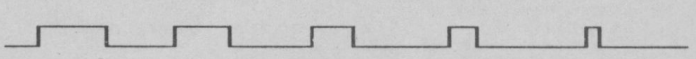


Рис 2. Приближение удара барабана

Если случайным образом чередовать 0 и 1, то получится белый шум. Сложных звуковых эффектов, таких как речь, получить не удавалось, но удары и шлепки получаются неплохо.

Программы, использующие этот способ, хорошо работают в фоновом режиме и не влияют заметным образом на скорость выполнения основной программы.

Следующий способ создания звука касается многоголосной музыки. Если складывать несколько периодических сигналов прямоугольной формы, то вывести их на динамик без искажений можно только в том случае, когда их периоды получены умножением минимального периода на целое число (рис. 3).

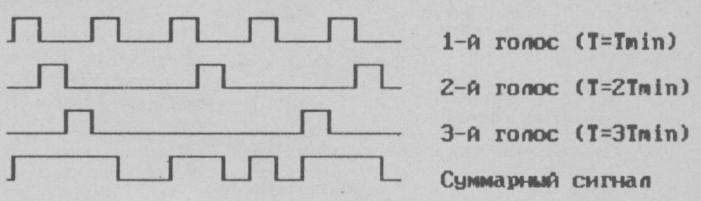


Рис. 3. Сложение сигналов с кратными периодами

То есть весь сигнал надо разбить на маленькие промежутки времени Tmin, и внутри каждого промежутка нужно выделить еще более мелкие отрезки, для

Programmer

Неотъемлемая часть Turbo Games — фоновые звуковые эффекты и музыкальное сопровождение

каждого голоса свои. В этом случае голоса не будут накладываться друг на друга. Максимальная частота звучания каждого голоса - 1/Tmin, а минимальное приращение периода - Tmin. Из-за скачкообразного изменения частоты при выводе музыки на высоких нотах появляется фальшь, чтобы избежать ее, надо следить за тем, чтобы в мелодии максимальная частота не превышала 1/(12*Tmin). Программа, использующая данный способ вывода, приведена в листинге.

В данном примере происходит вывод трехголосной музыки в фоновом режиме. Во время работы на экране переливаются разными цветами прямоугольники и звучит фрагмент из полонеза Огинского. При замене данных в массиве Melody будет выводиться любая другая музыка.

Во время работы программы перехватывается прерывание таймера. Обработчик прерывания и установщик этого обработчика находятся в теле одной процедуры TakeClock. Установщик сохраняет старый вектор прерывания таймера, устанавливает новый и передает адреса глобальных переменных в переменные, находящиеся в сегменте кода.

Через эти глобальные переменные в обработчик передаются значения, которые надо послать в 61H порт,— данные о музыке и число аккордов, которое надо сыграть.

Когда обработчик установлен, увеличивается частота генерации прерываний таймера. Это делается в процедуре SetSpeed, при этом частота прерываний становится равной 1193180/30 Гц. Переменной MuzRun присваивается значение TRUE.

С этого момента системные часы останавливаются и будут включены вновь только после окончания музыки, при этом они, конечно, отстанут на несколько секунд. Можно попробовать компенсировать этот недостаток, передавая управление раз в 65535 импульсов старому обработчику, но это приведет к некоторому усложнению программы.

Данные о музыке представляются четверками чисел, первые три из них - число периодов, которое будет пропускать каждый из голосов перед тем, как поднять мембрану динамика на один период. Последнее число - количество периодов звучания данного аккорда.

Когда аккорд заканчивается, число аккордов, которое нужно сыграть, уменьшается на единицу, и как только оно становится равным нулю, обработчик вызывает процедуру MuzClose, которая устанавливает низкую частоту прерываний таймера и возвращает вектор прерывания в исходное состояние. Восстанавливается также содержимое 61H порта. Переменной MuzRun присваивается значение FALSE, и это служит сигналом к окончанию работы программы.

Вообще говоря, звучание музыки можно прервать в любой момент, вызвав процедуру MuzClose из основной программы, не дожидаясь пока ее вызовет обработчик прерывания таймера.

Использование данного способа для создания музыки в фоновом режиме может сильно повлиять на скорость выполнения основной программы, если частота прерываний таймера будет высокой, а быстродействие машины небольшим. Поэтому лучше ограничить число голосов и сузить музыкальный диапазон исполняемого произведения, насколько это возможно.

Наиболее мощным и наиболее сложным является метод широтноимпульсной модуляции (ШИМ). Этот метод позволяет воспроизвести довольно точно практически любой звук, но для его реализации необходимо использовать аналого-цифровой преобразователь (АЦП).

Суть метода заключается в том, что можно управлять положением мембраны динамика за счет изменения ширины импульсов. Предположим, что мы подаем на динамик прямоугольные импульсы с постоянной частотой и переменной скважностью, причем частота импульсов превосходит максимальную частоту, воспроизводимую динамиком. В этом случае динамик будет усреднять протекающий через него ток и высота положения мембраны будет пропорциональна среднему току через динамик, то есть пропорциональна ширине импульсов.

Теперь осталось заставить таймер генерировать прерывания с частотой, превышающей на порядок верхнюю частоту динамика, и подключить эффективный обработчик этого прерывания, который успевал бы исправно подавать данные, полученные при помощи АЦП, на динамик.

Несмотря на то, что метод ШИМ требует огромных ресурсов времени, иногда встречаются программы, которые используют этот метод в фоновом режиме. ■

Монитор-1993

Подписной купон

Адрес _____

Ф.И.О. или название _____

организации _____

В 1993 году выйдут 10 номеров "Монитора".

Подписка: перечислите 1200 рублей на счет НТО "Софт-Москва": р/с 467002 в к/б "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г.Москве, МФО 201791; вышлите подписной купон и квитанцию об оплате по адресу: 117419, Москва, а/я 765.

Листинг

```
{$F+}
program List1;

uses Crt;

var
    Down: ^Byte;                    { Указатели на данные в сегменте кода }
    Up: ^Byte;                      {                                     }
    Music: ^Pointer;                {                                     }
    Cnt: ^Word;                     {                                     }
    OldTimeInt: Pointer;            { Указатель на старый вектор прерывания }
    OldPort, DinDown, DinUp: Byte;  { Данные для посылки в порт     }

const
    MuzRun: Boolean = FALSE;        { Признак того, что музыка играет}
    Count: Word = 80;               { Число аккордов                  }
    Melody: array [1..80,1..4] of Word = ( { Данные для музыки        }
    { Число минимальных периодов, которое нужно пропустить }
    { Iголос, IIголос, IIIголос, длительность аккорда }
    (99,41,32,2800), (41,32,66,5600), (34,99,34,1400), (99,32,32,1400)
    ,(41,99,31,2800), (32,99,32,2800), (99,66,41,2800), (66,41,41,1400)
    ,(66,44,44,1400), (66,49,49,2800), (49,49,83,2800), (99,49,49,2800)
    ,(66,41,41,2800), (41,32,99,2800), (99,32,32,2800), (99,41,24,2800)
    ,(24,32,41,2800), (24,24,78,2800), (32,32,55,2800), (74,27,27,2800)
    ,(74,29,29,2800), (49,31,31,2800), (31,74,31,2800), (31,62,49,2800)
    ,(36,36,2800,2800), (36,52,21,2800), (21,21,133,1400), (24,133,1400,1400)
    ,(26,26,133,2800), (26,26,66,1400), (66,31,1400,1400), (105,32,44,2800)
    ,(133,32,32,1400), (36,133,36,1400), (32,99,41,2800), (32,41,66,1400)
    ,(66,36,36,1400), (133,41,41,2800), (66,41,41,2800), (41,83,41,2800)
    ,(49,83,49,2800), (44,62,74,1400), (62,41,74,1400), (1400,44,44,1400)
    ,(49,49,1400,1400), (44,44,74,1400), (74,36,36,1400), (74,31,31,1400)
    ,(74,32,32,1400), (36,52,1400,1400), (41,1400,52,1400), (44,44,1400,1400)
    ,(1400,41,41,1400), (49,99,49,1400), (41,41,99,1400), (32,32,99,1400)
    ,(24,24,99,1400), (2800,20,20,2800),(66,20,20,2800), (99,20,20,2800)
    ,(32,41,24,2800), (21,31,74,1400), (74,31,20,1400), (21,31,74,1400)
    ,(24,74,31,1400), (21,21,74,1400), (74,18,18,1400), (15,15,74,1400)
    ,(16,16,74,1400), (1400,18,26,1400), (1400,26,20,1400), (21,21,1400,1400)
    ,(24,1400,24,1400), (26,26,1400,1400), (24,24,1400,1400), (21,21,1400,1400)
    ,(1400,26,26,1400), (36,26,21,2800), (36,26,21,2800), (32,24,41,2800)
    );

procedure SetSpeed (Speed: Word);   { Установить скорость прерываний }
begin                               { таймера                       }
    asm
        mov al, 00110110b
        out 43h, al
        mov ax, Speed
        out 40h, al
        mov al, ah
        out 40h, al
    end
end;

procedure MuzPutClock;              { Возвратить прерывание таймера    }
var
    TimeInt: Pointer absolute 0:8*4;

begin
    TimeInt := OldTimeInt;
end;

procedure MuzClose;                 { Привести все в начальное состояние }
begin
    SetSpeed (0);
    MuzPutClock;
    Port [$61] := OldPort;
    MuzRun := FALSE;
end;

procedure TakeClock;                { Перехватить и обработать      }
var                                 { прерывание таймера            }
    TimeInt: Pointer absolute 0:8*4; { Указатель на обработчик прерывания }
label                               {                               }
    ClckInt;                        {                               }
begin                               {                               }
    OldTimeInt := TimeInt;          { Сохранить старый вектор       }
asm                                 {                               }
    push    ds                      { Устанаовить свой обработчик   }
    mov     dx, offset ClckInt      {                               }
    mov     ax, cs                  {                               }
    mov     ds, ax                  {                               }
    mov     ax, 2508h               {                               }
    int     21h                     {                               }
    pop     ds                      {                               }
    mov     bx, cs                  { Переслать в глобальные переменные }
    mov     ax, offset @@Down       { указатели на переменные в сегменте }
    mov     word ptr Down, ax       { кода. После этого к ним можно }
    mov     word ptr Down[2], bx    { обращаться из других процедур}
    mov     ax, offset @@Up         {                               }
    mov     word ptr Up, ax         {                               }
    mov     word ptr Up[2], bx      {                               }
    mov     ax, offset @@Cnt        {                               }
    mov     word ptr Cnt, ax        {                               }
    mov     word ptr Cnt[2], bx     {                               }
    mov     ax, offset @@Music      {                               }
    mov     word ptr Music, ax      {                               }
    mov     word ptr Music[2], bx   {                               }
    jmp     @@Exit                  { Перепрыгиваем тело обработчика}
                                    { Данные для обработчика        }
@@Down:         db 0                { Динамик вниз                  }
@@Up:       db 0                    { Динамик вверх                 }
@@Cnt:      dw 0                    { Число аккордов                }
@@Music:    dd 0                    { Мелодия                       }
@@CurVoice: dw 0                    { Текущий голос                 }
@@VoiceCnt: dw 0                    { Число пропущенных периодов    }
            dw 0                    {                               }
@@VoicePro: dw 0                    { Длительность аккорда          }

ClckInt:                            { Собственно обработчик         }
    push ds                         {                               }
    push di                         {                               }
    push ax                         {                               }
    push    bx                      {                               }
    cli                             {                               }
    cmp     word ptr @@Cnt, 0       { Все сыграли?                  }
    jne     @@Nxt                   {                               }
    mov     ax, seg @Data           {                               }
    mov     ds, ax                  {                               }
    push es                         {                               }
    push si                         {                               }
    push cx                         {                               }
    push dx                         {                               }
    call  MuzClose                  { Закончить играть              }
    pop     dx                      {                               }
    pop     cx                      {                               }
    pop     si                      {                               }
    pop     es                      {                               }
    jmp     @@Done                  {                               }
@@Nxt:                              {                               }
    lds     bx, dword ptr @@Music   { В DS:BX - указатель на мелодию}
    mov     di, word ptr @@CurVoice     { Сдвинуть в DI номер голоса}
    shl     di, 1                   { Умножить его на 2 для смещения}
    mov     ax, word ptr @@VoiceCnt [di]    { Получить число }
                                            { пропущенных периодов }

    cmp     ax, word ptr bx [di]    { Сравнить его с данными        }
    je      @@DinUp                 { Если совпали - динамик вверх  }
{   @@DinDown: }
    inc     word ptr @@VoiceCnt [di] { Пропустить еще один период}
    mov     al, byte ptr @@Down     { Динамик опущен                }
    out     61h, al                 {                               }
    jmp     @@NxtVoice              { Следующий голос               }
@@DinUp:                            {                               }
    mov     word ptr @@VoiceCnt [di], 0 { Поднять мембрану динамика}
    mov     al, byte ptr @@Up       {                               }
    out     61h, al                 {                               }
@@NxtVoice:                         {                               }
    inc     word ptr @@CurVoice     { Следующий голос               }
    cmp     word ptr @@CurVoice, 3 { Если все сыграли,              }
    jne     @@Done                  {                               }
    mov     word ptr @@CurVoice, 0 {                                }
    inc     word ptr @@VoicePro     {                               }
    mov     ax, word ptr @@VoicePro {                               }
    cmp     ax, word ptr ds:bx [6]  { проверить конец аккорда       }
    jne     @@Done                  {                               }
    add     word ptr @@Music, 8     { Следующий аккорд              }
    dec     word ptr @@Cnt          { Уменьшить число аккордов      }
    mov     word ptr @@VoiceCnt [0], 0  { Сбросить число            }
                                        { пропущенных периодов      }
    mov     word ptr @@VoiceCnt [2], 0  {                           }
    mov     word ptr @@VoiceCnt [4], 0  {                           }
    mov     word ptr @@VoicePro, 0 { Сбросить длительность аккорда }
@@Done:                             {                               }
    mov     al, 20h                 { Сбросить контроллер           }
    out     20h, al                 {                               }
    pop     bx                      {                               }
    pop     ax                      {                               }
    pop     di                      {                               }

    pop     ds                      {                               }
    sti                             {                               }
    iret                            {                               }
@@Exit:                             {                               }
end;                                {                               }
end;                                {                               }

procedure MuzInit;                  { Привести все в готовность для }
begin                               { исполнения музыки             }
    if MuzRun then Exit;
    OldPort := Port [$61];
    DinDown := OldPort and $FE ;
    DinUp := DinDown or $02;
    asm cli end;
    TakeClock;
    Down^ := DinDown;
    Up^ := DinUp;
    asm sti end;
    MuzRun := TRUE;
end;

procedure Muz;                      { Подставить данные для музыки и }
begin                               { установить скорость            }
    MuzInit;
    Music^ := @Melody;
    Cnt^ := Count;
    SetSpeed (30);
end;

begin                                   { Основная программа       }
    ClrScr;
    Muz;                                { Запуск фоновой музыки    }
    repeat
        GoToXY (1 + Random (80), 1 + Random (24));
        TextColor (1 + Random (15));
        Write ('*');                    { Машина чем-то занимается, пока не }
    until not MuzRun;                   { кончится музыка          }
end.                                    { Конец.                   }
```

Коммерческие системы шифрования: основные алгоритмы и их реализация

"Каждый человек имеет право быть высокого мнения о себе, пока не станет знаменит..."
Какой-то философ

Часть 2. Системы с открытым ключом

В. Водолазкий

Итак, перед вами вторая часть статьи, посвященной коммерческим алгоритмам шифрования. В [1] мы рассмотрели алгоритм шифрования DES, а сегодня поговорим о системах с открытым ключом. Но прежде чем приступить к описанию конкретных алгоритмов, нам придется ненадолго обратиться к теории.

Однонаправленные функции

Несмотря на то, что вся концепция криптографических систем с открытым ключом основана на однонаправленных функциях (one way functions), достаточно сложно дать точное определение этого класса функций с математической точки зрения. Попробуем применить менее формальный, но зато более понятный подход. А именно , функция

$$f : x \to y,$$

где x и y - произвольные множества, является однонаправленной, если для всех x (E) X легко вычислить функцию f(x), и в то же время для большинства y (E) Y получить любое значение x (E) X, такое что f(x) = y достаточно сложно (при этом полагают, что существует, по крайней мере, одно такое значение x).

К сожалению, в настящее время математика не в состоянии дать нам ответ на вопрос, существуют ли таковые функции вообще или же это только красивая гипотеза. Тем не менее пытливым умам удалось обнаружить несколько зависимостей, которые могут быть использованы (и используются!) в качестве однонаправленных. Основной критерий причисления функции к классу однонаправленных очень прост - отсутствие эффективных алгоритмов обратного преобразования. Примеры? Пожалуйста!

1) Первый кандидат в однонаправленные функции - целочисленное умножение. В самом деле, вычислить произведение двух очень больших целых чисел (имеется в виду, с помощью ЭВМ, а не вручную) достаточно легко, но даже самый мощный компьютер с наилучшими известными на сегодняшний день алгоритмами не в состоянии факторизовать (разделить на сомножители) двухсотзначное число, которое является произведением двух сопоставимых по длине простых чисел.

2) Следующий характерный пример - это модульная экспонента с фиксированными основанием и модулем. Положим, что N и A - целые числа такие что

$$1 < A < N.$$

Далее, определим множество Zn:

$$Z = \{ 0, 1, 2, \ldots N-1\}.$$

Тогда модульная экспонента с основанием A по модулю N представляет собой функцию:

$$F_{a,n} : Z_n \to Z_n,$$
$$F_{a,n}(m) = A^m \text{ MOD } N. \qquad (1)$$

Существуют эффективные алгоритмы, позволяющие достаточно быстро вычислить значение функции по формуле (1). Обратная функция известна под названием задачи о дискретном логарифмировании: для известных целых A,N, X найти некоторое целое число M, такое что

$$A^m \text{ MOD } N = X$$

И хотя модульное возведение в степень может выполняться достаточно быстро, пока не удалось найти алгоритм, обеспечивающий вычисление дискретного логарифма за приемлемое время. В то же время не удалось и доказать, что такого алгоритма не существует. Исходя из этого, модульная экспонента отнесена к однонаправленным функциям условно, что, однако, не мешает с успехом использовать ее на практике.

Необходимо отметить, что любая однонаправленная функция (ОНФ) отнесена к этому классу как бы условно. Как показала практика, как только алгоритм получает достаточно широкое распространение (обращаясь к эпиграфу, становится знаменит), сразу же у определенных групп лиц возникает желание найти обратную функцию, и, поскольку это желание подкрепляется солидными денежными призами, не уверенные в себе кандидаты в ОНФ оказываются на помойке. Так что, читатель, у тебя есть возможность отличиться и посрамить апологетов буржуйских коммерческих систем шифрования.

Но вернемся к теме. Вторым важным классом функций, используемых в практике построения систем с открытым ключом, являются так называемые однонаправленные функции с черным ходом (trap door one way function). Для порядка введем определение.

Функция

$$f: X \to Y$$

относится к классу однонаправленных функций с черным ходом в том случае, если она является однонаправленной и, кроме того, возможно эффективное вычисление инверсной функции, если известен "черный ход" (или, говоря по-русски, секретная строка, число или другая информация, ассоциирующаяся с данной функцией).

"Достаточно запутанное определение", - подумает читатель и будет, как всегда, прав. Поэтому автор попробует пояснить сказанное примером.

В качестве демонстрации функции с черным ходом мы рассмотрим все ту же модульную экспоненту с фиксированными показателем и модулем. Пусть M и N - целые числа, а Z_n определено так же, как и раньше. Тогда модульная экспонента с фиксированным показателем M представляет собой функцию:

$$G_{m,n} : Z_n \to Z_n$$

или

$$G_{m,n}(A) = A^m \text{ MOD } n.$$

Для вычисления обратной функции нам необходимо извлечь из X корень M-й степени по модулю N, то есть для заданных целых чисел M, N и X найти некоторое целое число A, такое что

$$A^m \text{ MOD } n = X.$$

Если числа M и N известны, достаточно легко вычислить Gm,n(A) для любого основания A. Основное отличие от задачи вычисления дискретного логарифма состоит в том, что известен эффективный алгоритм вычисления корня m-й степени из X по модулю N.

Интересно отметить, что пока не известно, как эффективно реализовать этот алгоритм, если известны только M и N.

Завершая наше "теоретическое" введение, еще раз отметим, что любители математики и особенно "высшей арифметики" - теории чисел - имеют непочатый фронт работ как в области поиска новых однонаправленных функций, так и в области отыскания эффективных алгоритмов обратных преобразований. В случае решения любой из этих задач вы сможете забыть об обдираловке с импозантным названием "Лотто "Миллион". Миллионеры в эту "игру" не играют! За работу, господа!

Система распределения ключей Диффи–Хелмана

В традиционных криптографических системах (сразу вспоминается фильм "Семнадцать мгновений весны") каждая пара пользователей применяет один и тот же секретный ключ для шифрования и расшифровки сообщений.

Стоп! Давайте-ка определимся с терминологией. Не отрываясь от примера: Юстас - зашифровывает, Алекс - расшифровывает, а старина Мюллер - дешифрует сообщение. Теперь можно продолжать.

Это означает, что необходим надежный способ передачи ключа от одного пользователя к другому. Если пользователи меняют ключ достаточно часто, его доставка превращается в серьезную проблему. И более того, в традиционной криптографической системе просто невозможно передать информацию новому пользователю системы до тех пор, пока ему не будет по надежному каналу связи передан секретный ключ. И если спецслужбы как-то выходят из этой ситуации, то для коммерческих приложений это никуда не годится. И пытливая инженерная мысль нашла выход - была создана система распределения открытых ключей (public-key distribution system), позволяющая своим пользователям обмениваться секретными ключами по незащищенным каналам связи.

Первой системой такого рода стала система Диффи-Хелмана, разработанная в 1976 году, построенная на задаче о дискретном логарифмировании.

Предположим, что два пользователя, Алекс и Юстас, применяющие традиционную криптосистему, желают связаться друг с другом. Это означает, что они должны прийти к соглашению относительно ключа K, которым будут шифроваться сообщения. Давайте посмотрим, как система Диффи-Хелмана позволит обменяться ключом.

Пусть N - некоторое большое целое число, а G - другое целое, такое что

$$1 \le G \le N-1.$$

Рассмотрим процедуру обмена ключами по шагам.

1. Вначале Алекс и Юстас достигают соглашения о значениях N и G (как правило, эти значения являются стандартными для всех пользователей системы).

2. Затем Алекс выбирает некоторое большое целое число X и вычисляет

$$XX = G^X \text{ MOD } N.$$

Аналогичным образом Юстас выбирает число Y и вычисляет

$$YY = G^Y \text{ MOD } N.$$

После этого Алекс и Юстас обмениваются значениями XX и YY. (Мы считаем, что все данные, которые передаются по каналу связи, могут быть перехвачены злоумышленником - старинной Мюллером). Числа X и Y Алекс и Юстас хранят в секрете.

3. Получив от Юстаса число YY, Алекс вычисляет

$$K_1 = YY^X \text{ MOD } N,$$

а Юстас

$$K_2 = XX^Y \text{ MOD } N.$$

Но (!)

$$YY^X \text{ MOD } N = G^{X*Y} \text{ MOD } N = XX^Y \text{ MOD } N,$$

а следовательно,

$$K_1 = K_2 = K$$

Это значение K и является ключом, который используется для шифрования сообщений.

А что же старина Мюллер? Злоумышленник, перехвативший G, N, XX и YY, тоже должен определить значение ключа K. Очевидный путь для решения задачи состоит в вычислении значения X по G, N, XX или, по крайней мере, некоторого X^, такого что

$$G^{X^\wedge} \text{ MOD } N = X,$$

поскольку в этом случае

$$YY^{X^\wedge} \text{ MOD } N = K.$$

Однако это и есть задача дискретного логарифмирования в чистом виде, которая считается неразрешимой.

Система Диффи-Хелмана позволяет двум пользователям прийти к соглашению относительно общего секретного ключа. Однако система никак не влияет на то, как потом будет шифроваться сама информация. И если Алекс хочет передать Юстасу секретное сообщение M, то после установления ключа по Диффи-Хелману может быть использована любая система шифрования.

Но системы с открытым ключом создавались не только и даже не столько для решения задачи распределения ключей. При грамотном подходе возможно эффективное их использование для шифрования информации. Ведь, по определению, система с открытым ключом отличается тем, что тот, кто знает ключ для шифрования, не может дешифровать текст за практически приемлемое время.

Рассмотрим, как же используются системы с открытым ключом.

Пользователь Алекс имеет в своем распоряжении два алгоритма: E_a для шифрования и D_a для расшифровки сообщений. При этом алгоритм E_a делается общедоступным, например, через использование каталога ключей, а алгоритм D_a хранится Алексом в секрете. Если Юстас или даже старина Мюллер хочет послать Алексу сообщение, он ищет в каталоге ключей алгоритм E_a и использует его для шифрования передаваемой информации. А вот расшифровать сообщение сможет только Алекс, поскольку алгоритм D_a есть только у него. Очевидно, что E_a и D_a должны удовлетворять условию:

$$D_a(E_a(M)) = M$$

для любого сообщения M.

И снова, как и для традиционных криптосхем, требуется получить эффективные алгоритмы E_a и D_a. При этом необходимо, чтобы алгоритм E_a представлял собой функцию с черным ходом, то есть знание алгоритма E_a не должно быть достаточным для реализации D_a.

Системы с открытым ключом могут быть реализованы только в том случае, если подобрана однонаправленная функция с черным ходом. При этом необходимо постоянно помнить, что доказательства однонаправленности не существует. А из этого, в свою очередь, следует, что при выборе кандидатов в однонаправленные функции следует соблюдать известную осторожность, подкрепленную результатами тщательного тестирования.

Система RSA

Первую и наиболее известную систему с открытым ключом разработали еще в 1978 году трое американцев: Ривест, Шамир и Адлеман. И по первым буквам фамилий авторов система получила свое название - RSA. Основа системы - функция модульного экспоненцирования с фиксированными модулем и показателем степени.

Процедура шифрования-расшифровки в RSA реализуется следующим образом:

1. Выбираем два случайных больших простых числа P и Q и вычисляем

$$N = P * Q.$$

2. Выбираем большое целое число D, являющееся взаимно простым с произведением (P-1)*(Q-1) (для подзабывших арифметику напомним, что взаимно простыми являются два числа, наибольший общий делитель которых равен 1 [2]).

3. Вычисляем число e, такое что $1 \le e \le (P-1)*(Q-1)$ и

$$e*D = 1 \text{ MOD } (P-1)*(Q-1)$$

4. Обнародуем открытый ключ (алгоритм шифрования), состоящий из пары чисел (e,N).

5. Исходный текст (или отдельный символ сообщения) преобразуется в целое число M, такое что $0 \le M \le N-1$ (если текст слишком длинный, он разбивается на блоки соответствующей длины).

6. Текст шифруется в соответствии с алгоритмом:

$$C = M^e \text{ MOD } N.$$

7. Полученная криптограмма C расшифровывается с использованием личного ключа (D, N) по формуле:

$$C^D = M^{e*D} = M \bmod N.$$

Для того чтобы алгоритм RSA имел какую-то практическую ценность, необходимо иметь возможность без существенных вычислительных затрат генерировать большие простые числа, а также уметь вычислять значение ключа e. К счастью, такие алгоритмы существуют, а также разработаны быстрые алгоритмы для вычисления значения D.

Однако автор статьи не претендует на исчерпывающее описание всех тонкостей быстрого счета и приглашает читателей попробовать свои силы. Редакция готова опубликовать эффективные алгоритмы поиска простых чисел и, более того, объявляет конкурс, но об этом в конце статьи.

Теперь давайте рассмотрим пример реализации алгоритма RSA. Имейте в виду, что пример носит чисто учебный характер, хотя и отличается от "боевого" только малым (всего 2) числом разрядов в используемых ключах e и D.

Итак, положим, P=47, Q=59. Тогда N = P*Q = 2773, а (P-1)*(Q-1) = 2668.

Совершенно произвольно выберем D = 157. Вычислим e = 17. Теперь закодируем алфавит в виде:

А = 01, Б = 02, ... Я = 33, <пробел> = 00,

и тогда сообщение:

ОТБЫВАЕМ ЗАВТРА

будет представлено в виде :

1620 0229 0301 0614 0009 0103 2018 0100

Мы разбиваем все сообщение на блоки из четырех цифр и кодируем каждый блок. Например:

$$C_1 = 1620^{17} \text{ MOD } 2773 = 2316$$

В результате мы получим:

2316 2137 0214 1412 2072 0850 0985 1952

Рассмотрим теперь обратное преобразование. Реальная польза от любого алгоритма шифрования может быть получена только в том случае, если имеется возможность адекватно восстановить исходный текст. Применительно к системе RSA это означает, что для заданных публичного (e,N) и личного (D,N) ключей для любого сообщения M, представленного целым числом в диапазоне [0,N-1], должно выполняться соотношение:

$$C = M^e \text{ MOD } N,$$

тогда

$$C^D = M \text{ MOD } N.$$

То есть для восстановления исходного текста необходимо вычислить модульную экспоненту, возведя зашифрованное значение Ci в степень D по модулю N.

Замечания по применению

Будьте осторожны при выборе чисел P и Q. Наиболее общие правила: эти простые (!) числа не должны лежать рядом: одно из них должно быть больше другого на несколько порядков, P-1 и Q-1 должны иметь как можно меньший наибольший общий делитель. Поскольку на сегодняшний день не найден алгоритм, позволяющий за приемлемое время разложить на сомножители двухсотзначное целое число, вы можете считать ваш выбор удачным, если P и Q - простые числа со 100 десятичныи цифрами и более [3].

Необходимо также иметь в виду наиболее очевидные пути взлома системы RSA:

1) Факторизация N.

Как только злоумышленник сумеет разложить N на сомножители, он сможет вычислить F(N) = (P-1)*(Q-1), откуда легко определяется ключ D.

2) Вычисление F(N) без факторизации N.

Если определено значение F(N), то ключ расшифровки D вычисляется достаточно просто. Кроме того, знание F(N) позволяет легко разложить на сомножители число N. В самом деле:

$$P + Q = N - F(N) + 1$$
$$(P - Q)^2 = (P + Q)^2 - 4*N$$
$$Q = 1/2 * [(P + Q) - (P - Q)]$$

После этого без каких бы то ни было проблем вычисляется ключ D.

В помощь начинающим

Когда автор готовил пример, то для того чтобы получить реальные числа пришлось написать небольшую программу на Бейсике (выбор языка объясняется тем, что самураи из фирмы Sharp не догадались поставить в свой калькулятор PC-1600 хотя бы Паскаль - прим. автора).

Как ни странно, но эта программа (см. листинг) представляет собой определенный интерес, прежде всего, тем как организовано вычисление многоразрядных чисел. Поскольку быстрого алгоритма возведения в степень у автора не было, то применен алгоритм 6-го класса [2] - "перемножай в лоб". А поэтому :

* * * * * Конкурс! * * * * *

Редакция журнала "Монитор" объявляет конкурс на разработку лучшего алгоритма вычисления модульной экспоненты. Основные критерии : максимальная скорость вычислений и минимальная длина. Никаких ограничений на используемый язык программирования не предъявляется. Лучшие алгоритмы будут опубликованы на наших страницах.

Листинг

```
10   REM    ***********************************
20   REM          Вычисление модульной экспоненты
30   REM          (С) "Монитор" 1992
35   REM          Версия для PC-1600
40   REM    ***********************************
100  DIM T(100)
110  CLS
130  PRINT "Модульная экспонента"
140  CURSOR 1,2
150  PRINT "Введите модуль : ";
160  INPUT M
170  CURSOR 1,2 : PRINT "                    ";
180  CURSOR 1,2
190  PRINT "Степень : "; : INPUT P
210  CURSOR 1,2 : PRINT "                    ";
220  CURSOR 1,2
230  CURSOR 1,2 : PRINT "Число : "; : INPUT N
240  IF N = 0 THEN STOP
300  FOR I = 1 TO 20
310  T(I) = 0 : NEXT I : REM Очистка временного массива
320  H = 1 : REM Инициализация счетчика памяти
330  S = 1 : REM и счетчика шагов экспоненцирования
335  T(1) = 1
340  IF S = P THEN 500 :REM Если выполнены все умножения - выход
350  CURSOR 20,1 : PRINT S: REM Отображение текущего шага вычислений
352  Z = 0
420  FOR J = 1 TO 100
430  T(J) = T(J)*N+Z
440  IF T(J) < 10000 THEN 468
460  Z = T(J) \ 10000 : REM Целочисленное деление
465  T(J) = T(J) MOD 10000: GOTO 470
468  Z = 0
470  NEXT J
475  REM GOSUB 7000 : REM Отладочная печать промежуточного результата
480  S = S + 1: GOTO 340
500  REM GOSUB 7000 : REM Еще одна отладочная печать
510  REM  ----------------------
520  REM — Вычисление модуля числа —
530  REM  ----------------------
550  Z = 0
560  FOR V = 1 TO 100
570  Z = (T(101-V)+Z*10000) MOD M
580  NEXT V
590  R = Z
800  CURSOR 1,2: PRINT "                    ";
810  CURSOR 1,1: PRINT "Результат : ";R
820  G$ = INKEY$
822  IF G$ = "N" THEN GOTO 825 ELSE GOTO 820
825  CURSOR 1,1: PRINT "                    ";
830  GOTO 230
7000 REM Вывод на принтер очень длинного целого числа
7010 CSIZE 1
7020 FOR V = 1 TO 30
7030 LPRINT T(31-V);
7040 NEXT V: LPRINT
7050 RETURN
```


Литература

1. Монитор. - 1992. - N 6-7.
2. Математика: Учебник для 6 класса. - М.: Просвещение, 1991.
3. Trung Van Tran. Encipherment systems and autentication/ Доклад на конференции Kryptographie und Datensicherheit, 16.4 - 18.4.1991, Эссен, ФРГ.

Logitech®
MODULA-2
4.0

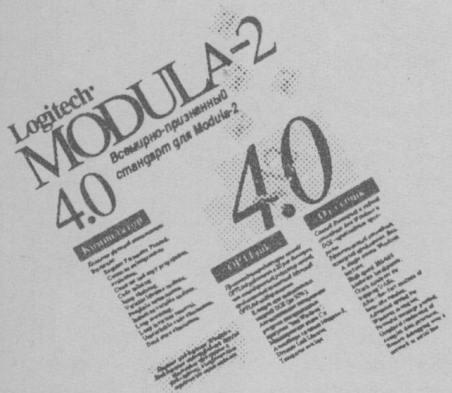
Всемирно-признанный стандарт для Modula-2

Компилятор

Комплекс функций оптимизации:
Включая:

- Register Parameter Passing;
- Common subexpression evaluation;
- Constant and copy propagation;
- Code inlining;
- Jump chaining;
- Variable lifetime analysis;
- Peephole optimization;
- Induction variable analysis;
- Loop rewriting;
- Loop invariant removal;
- Unreachable code elimination;
- Dead store elimination.

Полная поддержка Windows. Поддержка интерфейсов других языковых программ и библиотек, выпускаемых третьими компаниями.



OPTLink

Превосходные редакторы связей!
OPTLink работает в 20 раз быстрее, чем стандартный редактор связей, предлагаемый компанией Microsoft.
OPTLink поддерживает:

- Компрессию выполняемых модулей DOS (до 50%);
- Автоматическое копирование ресурсов для прикладных программ "под Windows";
- Микширование кодов С и Ассемблера с кодами Модулы-2;
- Dynamic Link Libraries;
- Transparent overlays.

По вопросам оптовых закупок компилятора Modula-2 4.0 обращайтесь к дистрибуторам Symantec:
Перспективные Технологии
тел. 256-4242, 256-4030
Трио Плюс
тел. 281-0375, 971-1204
Merisel- Компьютерные Технологии
тел. 276-9008
Диалог-МИФИ
тел. 320-3466, 320-4366
Все компании находятся в Москве

Отладчик

Самый быстрый и гибкий отладчик для Windows и DOS-прикладных программ
Единственный отладчик, который предлагает Вам:

- A single screen Windows intrface;
- High speed 386/486 hardware breakpoints;
- Crash Analyzer for trapping UAEs;
- After-the-fact analysis of beta-site crashes;
- Advanced spying for Windows messgaes;
- Powerful message posting;
- Graphical rendereing of complex data structures;
- Remote debugging over a network or serial line

SYMANTEC

От программиста до конечного пользователя: программные продукты Computer Associates

Д. Фивейский

Фирма Computer Associates (CA) - крупнейший независимый производитель программного обеспечения. Она предлагает более 300 программных решений для больших, средних и мини-ЭВМ и более 30 программных продуктов для персональных компьютеров. Computer Associates специализируется на создании программных продуктов для административных и экономических приложений.

Система разработки приложений CA–Clipper

Первая версия программного продукта Clipper - компилятора с языка dBASE - появилась в 1985 году. Все версии Clipper созданы по принципу открытой архитектуры, то есть пользователю предоставляются средства, позволяющие модифицировать систему в соответствии со своими потребностями. В последней версии Clipper 5.0, появившейся в 1990 году, такими средствами являются пользовательские функции, препроцессор и система расширения, обеспечивающая передачу параметров между функциями, написанными на Clipper, Си и ассемблере. Важными нововведениями, которые сделали Clipper профессиональным средством создания приложений, явились администратор виртуальной памяти, новый компоновщик RTLink, позволяющий использовать при выполнении программ динамический оверлей, и отладчик.

Будущая версия Clipper будет иметь компилятор в машинный код, что позволит достичь уровня производительности программ на языке Си. Она станет полной и замкнутой объектно-ориентированной системой. В составе системы появится интегрированная среда, которая предоставит визуальный инструментарий разработки. Совместно с объектной ориентацией он должен облегчить программирование графических пользовательских интерфейсов.

Библиотека функций CA–Clipper Tools II

В начале 1988 года была выпущена специализированная библиотека инструментальных функций Clipper Tools One, охватившая широкий диапазон функций,таких как поддержка окон, связь через последовательный интерфейс, обработка строк и многое другое. Необходимость включения функций, относящихся к работе в локальных сетях и к дополнительным режимам формирования видеоизображений, привела к созданию в начале 1989 года инструментальной библиотеки второй очереди. CT II включает все функции CT I, а также обеспечивает доступ более чем к 200 новым функциям. При этом соблюдается совместимость "снизу вверх".

Табличный процессор CA–SuperCalc

Электронные таблицы впервые появились в начале 80-х годов (VisiCalc и Multiplan). К этому же времени относится появление первой версии табличного процессора SuperCalc. Важнейшим преимуществом CA-SuperCalc была и остается простота освоения и использования. Последняя версия CA-SuperCalc 5.5, появившаяся в 1992 году, включает поддержку интерфейса при помощи "мыши", трехмерное связывание целых таблиц и их отдельных страниц, находящихся как в памяти, так и на диске, подготовку высококачественных отчетов с предварительным просмотром и автоматическим масштабированием при выводе, создание графиков более чем 100 различных типов, работу в локальной сети. Макроотладчик позволяет находить ошибки в макросах, выполнять их в пошаговом режиме. Кроме того, CA-SuperCalc 5.5 позволяет подключать приложения. Одним из них является реляционная база данных Silverado, обеспечивающая сортировку, поиск, выделение и фильтрацию данных.

Будущая версия CA-SuperCalc для Windows будет иметь полную реализацию графического пользовательского интерфейса, настраиваемое окно операций, возможность встраивания графиков и отформатированного текста в электронные таблицы. Язык программирования - расширенная версия языка Бейсик (CA-Basic Language).

Система управления проектами CA–SuperProject

Методы и алгоритмы сетевого планирования (метод критического пути, ПЕРТ) наиболее удобно воплощаются в системах управления проектами для персональных компьютеров. Программы управления проектами входят в пятерку лидеров по объему продаж. Их использование приносит реальную прибыль: облегчает процессы планирования и управления, позволяет рационально использовать ресурсы и сокращать сроки выполнения работ.

CA-SuperProject позволяет составлять расписание выполнения операций, то есть определять сроки начала и завершения как отдельных операций, так и проекта в целом, резервы времени, имеющиеся у отдельных операций проекта, выводить графики загрузки и использования ресурсов. Возможно задание иерархической структуры как для операций (фазы, подпроекты), так и для ресурсов, а также оперативное изменение плана с учетом изменяющихся обстоятельств и проведение оценок "что-если". CA-SuperProject предлагает разнообразные формы вывода: сетевые диаграммы, диаграммы Гантта, гистограммы загрузки ресурсов и распределения затрат во времени. Имеется также возможность обмена данными с CA-SuperCalc.

CA-SuperProject для Windows обеспечивает все возможности графического пользовательского интерфейса, вывод на экран одновременно различных представлений одного и того же проекта или нескольких разных проектов. Кроме того, он позволяет создавать высококачественные отчеты в режиме полного соответствия.

Заключение

Даже такой далеко не полный перечень программных продуктов Computer Associates позволяет сделать вывод о том, что фирма не ограничивает круг своих пользователей какой-либо одной категорией. Она предлагает программные решения как для профессиональных программистов, так и для менеджеров разного уровня и всех пользователей персональных компьютеров. Программные продукты фирмы поддерживают различные платформы. Сходный интерфейс, совместимость по форматам данных позволяют совместно использовать несколько программных продуктов, добиваясь наиболее эффективного решения задач. ■

Проблема памяти и адаптер VGA

М. Иринчук

Что делать, если программе не хватает памяти? Наверно, такой вопрос возникал у любого программиста при разработке программных комплексов, требующих хранения большого объема информации и быстроты доступа к ней.

Проблема памяти всегда оставалась ключевым моментом для разработчиков аппаратного обеспечения, операционных систем, прикладных программ, и решали они ее по-разному. Роковой 640-килобайтовый барьер DOS преодолевался за счет использования свободных участков в UMB (Upper Memory Blocks), не занятых ROM, и HMA (High Memory Area), путем загрузки в нее резидентных программ и драйверов. Именно эти два типа памяти использует знаменитая MS-DOS 5.0, размещая в ней ядро операционной системы и предоставляя прикладным программам до 590 килобайтов свободной памяти.

Барьер мегабайта преодолевался за счет использования EMB памяти (Extended Memory Blocks) (рис. 1), размер которой зависит от конкретной конфигурации машины. Эта память чаще всего используется для организации виртуальных дисков, что очень удобно при работе с временными файлами. Дополнительная память может быть размещена на отдельной плате, так называемой Expanded memory, работу с которой также поддерживает BIOS. Но речь пойдет не об этой памяти! Ведь платы Expanded memory может и не оказаться, а попытки использовать свободные участки в HMA или в UMB способны повредить ядро операционной системы или же код резидентной программы и привести к зависанию системы. Использование же Extended memory возможно лишь под управлением какого-нибудь менеджера памяти, типа HIMEM.SYS, так как эта память доступна только при работе процессора в защищенном режиме, который не поддерживает DOS. Так где же искать память, которая всегда доступна при работе с любимой DOS?

Не отчаивайтесь, выход есть! Если Вы разрабатываете программное обеспечение, работающее в алфавитно-цифровом режиме, и обладаете видеоадаптером EGA или VGA с 256 Кбайтами на плате, а именно о такой конфигурации пойдет речь дальше, то от 184 до 240 Кбайтов можно найти на самой видеоплате.

Всем известно, что в алфавитно-цифровых режимах видеобуфер начинается с сегментного адреса B800 (рис. 1), а в графических с A000, но, наверно, не все представляют, как можно использовать сегмент A000 в текстовом режиме. Для того чтобы ответить на этот вопрос, разберем основные аспекты физической организации адаптера.

Память видеоадаптера организована в виде четырех битовых матриц (bit planes), каждая размером в 64 Кбайта. Наглядно битовые матрицы можно представить в виде блоков памяти, расположенных один над другим и имеющих одни и те же адреса памяти. Такая организация памяти довольно удобна при программировании, а главное, позволяет сэкономить адресное пространство процессора. Адаптер предусматривает различные методы записи данных, поступающих от процессора в память, и чтения из нее, а также различные способы адресации, зависящие от установленного режима, числа используемых цветов и так далее. Нет необходимости рассматривать более детально работу адаптера, так как уже существует немало публикаций, досконально разбирающих физическую организацию адаптера и методы его программирования. Однако все они касались только работы адаптера в графическом режиме и скромно умалчивали о работе в алфавитно-цифровом. Попытаемся ликвидировать этот недостаток.

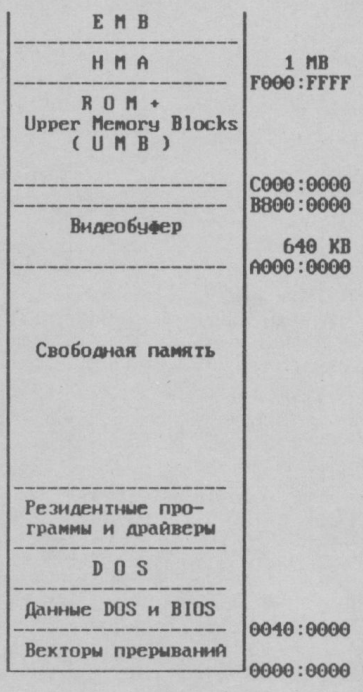


Рис 1. Распределение памяти в операционных системах DOS

В целях обеспечения совместимости с режимами CGA адаптер использует метод нечетной/четной адресации. В нечетно/четном режиме четные адреса памяти записываются в четные битовые плоскости, а нечетные адреса - в нечетные. Таким образом, работая в алфавитно-цифровом режиме, ASCII код символа помещается в плоскость 0, а его атрибут в плоскост 1. В целях простоты адресации адаптер корректирует эти адреса таким образом, что информации о символе и его атрибуте соответствует один и тот же адрес в адаптере. Сразу следует отметить, что при установке алфавитно-цифрового режима адаптер маскирует битовые плоскости 2 и 3, и запись в них не производится.

Известно, что если по адресу B800:0000 поместить ASCII код символа, а по адресу B800:0001 его атрибут, то символ с указанным цветом появится в левом верхнем углу экрана. Схема записи данных из ЦП в память адаптера представлена на рис. 2.

Таким образом, битовая плоскость 0 используется для хранения ASCII кодов выводимых символов, а плоскость 1 - для их атрибутов. А теперь оценим объем памяти, требуемый для хранения всех восьми видеостраниц (размером 80x25), поддерживаемых алфавитно-цифровыми режимами. Нетрудно убедиться, что в каждой из битовых плоскостей 0 и 1 занятыми оказываются менее 16 Кбайтов из 64, да и то если мы собираемся использовать все восемь страниц экрана. Оставшаяся же память этих плоскостей ничем не занята и может быть использована для любых целей. Однако здесь существует небольшая тонкость. Дело в том, что запись экранных символов и их атрибутов происходит только в четные адреса 0 и 1 плоскостей. Та-

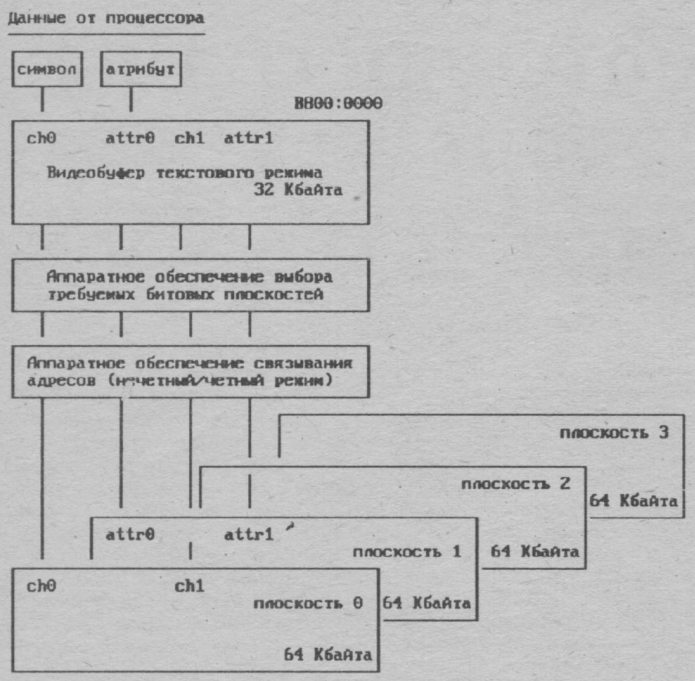


Рис. 2. Схема записи данных от ЦП в память адаптера

ким образом, в первых 32 Кбайтах обеих матриц свободные байты имеют нечетные адреса и, видимо, мало пригодны для использования.

Теперь разберем назначение оставшихся двух битовых плоскостей. Сразу скажу о битовой плоскости 3, она целиком свободна и может использоваться в Ваших прикладных программах. А вот с плоскостью 2 дело обстоит немного сложнее. Эта плоскость предназначена для хранения фонтов отображаемых символов. Первые восемь килобайтов этой плоскости имеют следующий вид: массив из 256 (полный набор ASCII) структур типа:

```
Font_T    Struc
          Char1 db 16 dup (?)
          Char2 db 16 dup (?)
Font_T    EndS
```

Такое описание справедливо для ассемблера; поскольку не все знакомы с этим языком программирования, разберем его более подробно. Каждый элемент N массива из 256 элементного массива представляет собой два массива по 16 байтов каждый: Char1 и Char2. Массив Char1 - основной вариант кодировки символа N, а массив Char2 - альтернативный. Отсюда становится понятным такой часто используемый термин, как два набора символов ASCII. Длина 16 байтов выбрана, так как для VGA размер символа 8x16, если же Вы используете фонты размером 8x8, то задействованными будут только первые восемь байтов из каждого массива.

Если Вашей прикладной программе требуется часто изменять фонты некоторых символов, то прямая модификация соответствующих элементов этих массивов значительно предпочтительней, чем использование соответствующих функций BIOS. Именно таким путем в алфавитно-цифровом режиме создается плавно перемещающаяся стрелка мыши, какую Вы наверняка видели во многих известных программных продуктах.

Итак, если Вашей прикладной программе достаточно двух наборов символов ASCII, то оставшиеся 56 Кбайтов этой плоскости в Вашем распоряжении.

И, наконец, можно подвести итоги, касающиеся неиспользованных ресурсов памяти. Если Вы используете одну страницу видеопамяти размером 80 на 25 , то в Вашем распоряжении 62+62+56+64=244 Кбайта. Если используете все восемь страниц, то 32+32+56+64=184 Кбайта.

Теперь, разобрав назначение всех четырех битовых матриц и оценив размеры неиспользуемых ресурсов памяти, перейдем к самой интересной части этой статьи: как использовать эту память в Ваших прикладных программах.

Ранее упоминалось, что речь пойдет о сегменте A000, - именно через эти адреса можно получить доступ ко всему пространству битовых плоскостей. Итак, процесс доступа к адресам i-й битовой матрицы происходит в три этапа.

1. Открываем битовую плоскость I вызовом функции OpenPlane(i);
2. Выполняем любые операции с требуемыми адресами сегмента A000;
3. Закрываем плоскость вызовом функции ClosePlane.

Ниже приводятся тексты функций OpenPlane(i) и ClosePlane, они могут быть скомпонованы с Вашей программой на C или C++ без каких-либо изменений. Для связывания с языком Pascal Вам необходимо заменить команду Ret в функции OpenPlane на команду Ret 2 (листинг 1).

И в заключение привожу небольшую программу, подтверждающую достоверность изложенного материала (листинг 2). Она производит очистку экрана с использованием рассмотренного метода доступа к адресному пространству адаптера и рисует во второй строке экрана две буквы F. Разумеется, данная программа не является оптимальной, с точки зрения всех канонов языка ассемблера, и служит лишь наглядной иллюстрацией материала статьи. ■

Листинг 1

```
ClosePlane      Proc Far
  Mov     Dx,3C4   ;Выбрать регистр указателя последовательности
  Mov     Ax,0302  ;Маскировать плоскости 2 и 3
  Out     Dx,Ax    ;
  Mov     Ax,0304  ;Установить нечетно/четный режим адресации
  Out     Dx,Ax    ;
  Mov     Dx,3CE   ;Выбрать регистр адреса графики 1 и 2
  Mov     Ax,1005  ;Установить нулевой режим чтения и записи
  Out     Dx,Ax    ;и нечетно/четный режим адресации
  Mov     Ax,0E06  ;Установить карту памяти (B800 / 32Кбайта) и
  Out     Dx,Ax    ;нечетно/четный режим адресации
  Mov     Ax,0004  ;Выбрать схему чтения 0 (номер плоскости)
  Out     Dx,Ax    ;
  Ret
ClosePlane      EndP
OpenPlane       Proc Far
  Plane = [Bp+6]
  PushBp
  Mov     Bp,Sp
  Mov     Dx,3C4   ;Выбрать регистр указателя последовательности
  Mov     Cl,Byte Ptr Plane
  Mov     Ax,0102
  Shl     Ah,Cl
  Out     Dx,Ax    ;Маскировать все плоскости, кроме Plane
  Mov     Ax,0704  ;Запретить нечетно/четный режим адресации
  Out     Dx,Ax
  Mov     Dx,3CE   ;Выбрать регистр адреса графики 1 и 2
  Mov     Ax,0005  ;Установить нулевой режим чтения и записи
  Out     Dx,Ax    ;и запретить нечетно/четный режим
  Mov     Ax,0406  ;Установить карту памяти (A000 / 64 Кбайта)
  Out     Dx,Ax    ;и сбросить нечетно/четный режим
  Mov     Ah,Byte Ptr Plane
  Mov     Al,04
  Out     Dx,Ax    ;Выбрать схему чтения Plane
  Pop     Bp
  Ret
OpenPlane       EndP
```

■

Листинг 2

```
.Radix 16
Code Segment
        Assume Cs:Code,Ds:Code
  Start:
        Cld
        ;--- Открыть плоскость 0
        Push0
        Call OpenPlane
        Add     Sp,2
        ;--- Заполнить весь экран пробелами
        Mov     Ax,0A000
        Mov     Es,Ax
        Xor     Di,Di
        Mov     Al,' '
        Mov     Cx,80d*50d
        Rep     Stosb
        Mov     Di,160d
        ;--- Вывести две буквы F во второй строке
        Mov     Byte Ptr Es:[Di],'F'
        Mov     Di,162d
        Mov     Byte Ptr Es:[Di],'F'
        ;--- Открыть плоскость 1
        Push1
        Call OpenPlane
        Add     Sp,2
        ;--- Установить атрибут 34 для всех символов экрана
        Xor     Di,Di
        Mov     Al,34
        Mov     Cx,80d*50d
        Rep     Stosb
        ;--- Установить атрибуты букв F
        Mov     Di,160d
        Mov     Byte Ptr Es:[Di],35
        Mov     Di,162d
        Mov     Byte Ptr Es:[Di],53
        ;--- Закрыть плоскость
        Call ClosePlane
        Mov     Ax,4C00
        Int     21
  Code  EndS
    End Start
```

■

Замедление работы программ

Д. Правиков

Одним из достоинств программ всегда являлась высокая скорость их выполнения, причем как идеал рассматривался реальный масштаб времени. Тем не менее иногда, особенно при использовании старого программного обеспечения на новых мощных компьютерах, бывает необходимо искусственно замедлить скорость выполнения программ, как правило, игровых, создатели которых не рассчитывали на столь быстрый и значительный прогресс вычислительной техники. Для решения этой проблемы можно предложить три способа.

Первый способ основан на непосредственном управлении скоростью центрального процессора на машинах Compaq 386. Функция 0F0h прерывания 16h позволяет варьировать скорость процессора, используя следующие подфункции:

AL 00h - установить эквивалент 6 МГц 80286;
01h - установить эквивалент 8 МГц 80286;
02h - установить эквивалент 16 МГц;
03h - переключение между эквивалентом 8 МГц и скоростью, задаваемой переключателями на системной плате;
08h - полные 16 МГц, за исключением эквивалента 8 МГц при доступе к гибким дискам;
09h - непосредственное задание скорости;
CX - номер скорости от 1 до 50.

К сожалению, данный способ ограничен только одним типом машин, поэтому не годится для широкого применения. Более общими являются оставшиеся два способа.

Один из этих способов основан на обработке прерывания от таймера, инициируемого 18,2 раза в секунду. Это аппаратное прерывание делит время работы центрального процессора на кванты, причем меньшая часть кванта используется для обработки прерывания от таймера, а оставшаяся большая часть - для работы системных и прикладных программ. Идея замедления в этом случае очень проста. Увеличение доли времени на обработку прерывания от таймера приводит к уменьшению доли на полезную работу. В настоящее время существует целый ряд программ замедления, основанных на этом принципе.

Гораздо более интересным способом, с точки зрения системного программирования, является замедление с помощью пошагового выполнения программ. Принцип замедления в этом случае основан на том, что при взведенном T флаге центрального процессора после выполнения каждой команды программы генерируется прерывание, обработка которого позволяет замедлить работу программы. Говоря другими словами, программа замедляется оттого, что она выполняется как бы под отладчиком. В листинге приведен исходный текст программы, позволяющей загрузить определенный COM файл, выполнить его в пошаговом режиме и вернуть управление в основную программу. В целом данная программа довольно близка к примеру в [1], иллюстрирующему запуск дочернего процесса, но вместе с тем имеет ряд отличий, на которые хотелось бы обратить ваше внимание.

Основное отличие заключается в том, что вместо подфункции "загрузить и выполнить" (AL=0) используется другая подфункция (AL=1), которую можно охарактеризовать как "загрузить, но не выполнить". Вследствие использования этой подфункции приходится дополнительно решать задачи оформления адреса возврата и определения точки входа в дочернюю программу. Оформление адреса возврата осуществляется следующим образом. В PSP (program segment prefix) дочерней программы, сегментный адрес которого можно получить с помощью функции 62h DOS (обратите внимание: эта функция возвращает сегментый адрес PSP последней загруженной а не текущей программы), адрес завершения корректируется так, чтобы он указывал на нужную точку в родительской программе. Этого достаточно для того, чтобы по завершении дочернего процесса управление вернулось в родительский процесс.

Определение точки входа в дочернюю программу описано в [2] следующим образом. Перед вызовом подфункции "загрузить, но не выполнить" в регистры ES:BX помещают адрес блока параметров (подробнее назначение и содержимое этого блока см. [1]). После выполнения функции в блок параметров помещается адрес точки входа в программу, а регистры ES:BX должны указывать опять на этот блок. В действительности же они на него не указывают, поэтому, чтобы получить адрес точки входа в программу, необходимо регистр BX снова установить на блок параметров.

Имея адрес точки входа, нетрудно передать программе управление. Для этого сначала переустанавливаются регистры стека, а затем в новый стек помещаются сначала флаги, затем сегмент и смещение точки входа, после чего дается команда возврата из прерывания. Если в стеке модифицировать слово флагов так, чтобы на месте T флага стояла 1, то дочерняя программа будет выполняться в пошаговом режиме. Единственная тонкость в этом месте - надлежащим образом оформить сегментные регистры ES и DS.

В заключение хотелось бы сделать несколько замечаний. Приведенная в листинге программа максимально упрощена, но вполне достаточна, чтобы замедлить до нормального уровня популярную игру Digger на машинах с 286 процессором. Дальнейшее развитие этой программы можно продолжить в следующих направлениях. Во-первых, добавить обработку 01h прерывания (то есть увеличить время выполнения каждой команды) для значительного замедления на более мощных машинах. В приводимом варианте считается, что вектор 01h указывает на команду iret. Во-вторых, следует ввести обработку командной строки, и в-третьих, чтобы сделать программу более универсальной, можно добавить небольшую ветку для правильного оформления сегментных регистров при загрузке EXE файлов. ■

Листинг

```
; Программа замедления игры Digger
;
        .MODEL  SMALL
        .STACK          400h
        .DATA
filename        db   'digger.com',0
parameters      dw   20 dup(0)
keep_ss             dw   ?
keep_sp             dw   ?
        .CODE
Start:  mov     ax,@data
        mov     ds,ax
;------------------------------ Перераспределение памяти
        mov     bx,ZSEG
        mov     ax,es
        sub     bx,ax
        mov     ah,4Ah
        int     21h
        push ds
        pop     es
        mov     bx,OFFSET parameters  ; BX -> блок параметров
;------------------------------ Сохранение указателя стека
        mov     keep_ss,ss
        mov     keep_sp,sp
        mov     dx,OFFSET filename          ; DX -> имя файла
        mov     ax,4B01h                    ; Загрузка, но не выполнение
        int     21h
;------------------------------ Установка адеса возврата
        mov     ah,62h
        int     21h
        mov     ds,bx
        lea     bx,Continue
        mov     ds:[0Ah],bx
        mov     ds:[12h],cs
        mov     bx,OFFSET parameters
;------------------------------ Переустановка стека
        mov     ss,es:[bx+10h]
        mov     sp,es:[bx+0Eh]
;------------------------------ Установка T флага
        pushf
        pop     ax
        or      ax,100h
        push ax
        push    es:[bx+14h]
        push    es:[bx+12h]
;------------------------------ Для COM программ CS = DS = ES
        mov     ds,es:[bx+14h]
        mov     es,es:[bx+14h]
        xor     ax,ax
        xor     bx,bx
        iret
;------------------------------ Возврат из дочерней программы
Continue:
        mov     ax,@data
        mov     ds,ax
        mov     ss,keep_ss
        mov     sp,keep_sp
        mov     ax,4C00h                    ; Завершение программы
        int     21h
ZSEG segment
ZSEG ends
        END Start
```

■

Литература

1. Р. Джордейн. Справочник программиста персональных компьютеров типа IBM PC, XT и AT. - М.: Финансы и статистика, 1992. - С.47-51.
2. Dave Williams. Programmer's Technical Reference for MS-DOS and IBM PC. 1989.

ЭКСИМЕР

ЭКСИМЕР представляет новую продукцию:

- Персональные компьютеры серии **486 SX**. Возможности и быстродействие 486-й модели, стоимость ниже 386-й.
- Компактная, высокоскоростная, современная РАБОЧАЯ СТАНЦИЯ **KAIYO®** SL-150-процессор 80386 SX/33, корпус Super Slim. Лучшее решение для компьютерных сетей и индивидуальных работ.
- **Windows® accelerator**-видеоадаптер, ускоряющий обмен информацией в 15-20 раз.Сложнейшие графические задачи-доступны каждому.
- **JM-keyboard**-клавиатура в стандартном исполнении, позволяющая выполнять все операции манипулятора "мышь", функции системы "Windows"® простым нажатием клавиши.
- Сетевое и телекоммуникационное оборудование, в том числе продукт года в США- FAX/MODEM/VOICE **ZyXel®**.
- Различные аксессуары: держатели для документов, "боксы" для хранения дискет, защитные экраны всех типов.

Текущие цены на компьютеры сборки "ЭКСИМЕР":

386 DX/387/33	
64 Кб/4 Мб/120 Мб/SVGA 512 Кб	**$ 1386**
486 DLC/33	
64 Кб/4 Мб/120 Мб/SVGA 512Кб	**$ 1466**
486 DX2/50	
Local Bus SVGA 1 Мб/4 Мб/120 Мб	**$ 1972**
486 DX2/66	
Local Bus SVGA 1 Мб/4 Мб/120 Мб	**$ 2144**

— гарантия 1 год,
— возможность оплаты в рублях,
— скидка для оптовых покупателей до 10 %.

Телефон:(095) 125-70-01 Факс: (095) 125-78-68
Адрес: 117292, Москва, ул. Ивана Бабушкина, д. 23, корп. 1, метро "Профсоюзная"

Можете ли Вы с ходу назвать хотя бы три отечественные фирмы, выпускающие IBM PC совместимые ЭВМ? А если усугубить вопрос и спросить о машинах с 386 процессором? А с 486? А с 20 моделями ЭВМ с i386 и i486 на потоке?

Есть такая фирма! "Эксимер", заключив дистрибьюторское соглашение с фирмой Western Digital (США) на поставку жестких дисков, начиная с октября 1992 года, производит, то есть собирает, тестирует, проводит гарантийное обслуживание (1 год) 20 моделей IBM PC совместимых машин. Откуда такое многообразие? Все очень просто. "Эксимер" старается дать любому покупателю возможность приобрести именно то, что нужно, а не то что есть. Судите сами, фирма поставляет машины, собранные на базе одной из шести модификаций:

386/SX 33 Мгц, 386/DX + 387/DX 33 Мгц, 386/DX + 387/DX 40 Мгц, 486/DLC 33 Мгц, 486/DX2 50 Мгц Local bus/SVGA, 486/DX2 66 Мгц Local bus/SVGA.

К каждой из этих плат Вы можете подобрать винчестер (интерфейс IDE) емкостью 80, 120, 170, 210 или 340 Мбайтов. (Кстати, гарантия на жесткий диск по договору с фирмой Western Digital составляет 2 года!) Дополняют стандартный комплект ОЗУ 4 Мбайта, FDD's 1,2/1,44 Мбайта, адаптер SVGA (512 Кбайтов) и монитор с разрешением 1024 x 768 точек, 256 цветов. Все модели (еще раз повторим, что речь идет о серийных изделиях, а не о заказных уникальных комплектах) аккуратно оформлены в корпус Mini Tower. Но при желании Вам могут поставить и более солидный Medium Tower.

Так что покупатель, попавший в "Эксимер", оказывается перед мучительной, но в то же время приятной проблемой выбора - что купить? И может приобрести то, что хочется, а не то, что есть. А это уже немало.

Если же Ваши задачи требуют использования нестандартной периферии либо огромного объема памяти, "Эксимер" поможет Вам и в этом случае. Фирма предлагает широкий набор комплектующих узлов и устройств, устанавливаемых в приобретаемый Вами комплект или поставляемых отдельно. Упомянем только об одном изделии - факс модеме ZyXeL (США). Это самая популярная модель года, выполнена в виде отдельного устройства и обеспечивает прием/передачу факс почты со скоростью до 14400 (протокол V17) бод и обмен данными со скоростью 16800 бод/MNP5/V42bis.

В заключение заметим, что благодаря грамотно поставленной ценовой политике, а также наличию отличной ремонтной базы (ремонтируются, кстати, не только ЭВМ своей сборки) фирма "Эксимер" не испытывает трудностей с заказами и сбытом. Торопитесь, иначе придется стоять в очереди!

ЭКСИМЕР может: многое-для всех и единственное-для Вас!

Укрощение строптивого, или двадцать четыре вместо девяти

Часть 2

Н. Косарев

"Осмотрите конверт повнимательнее, и вы увидите, что писавшему не повезло с письменными принадлежностями."
А. Конан Дойл. Собака Баскервилей.

В предыдущем номере "Монитора" в статье "Копия экрана? Это очень просто!" был предложен модуль Screen.pas, дающий возможность получения графической копии экрана или его участка для адаптеров CGA,EGA,VGA. При этом подразумевалось наличие 9-игольчатого принтера. Если использовать 24-игольчатый принтер, то полученная на нем копия экрана будет по качеству уступать предыдущей, ибо Screen эмулирует на 24-игольчатом принтере обычный 8-игольчатый графический режим.

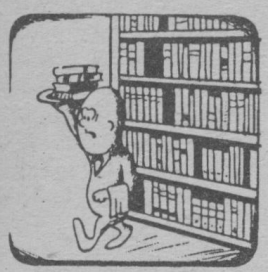
Для того чтобы заставить работать все 24 иглы, используем следующее решение, в основе которого лежит тот факт, что в принтер можно загрузить определяемый пользователем шрифт. Здесь необходимо сделать замечание. 24-игольчатые принтеры разных фирм-изготовителей имеют отличающиеся диапазоны номеров символов, которые можно переопределить. Так, например, модель "Amstrad LQ 3500 di 24 pin" дает возможность загрузить шрифт в первую половину кодировочной таблицы (от 0 до 127), модель Star XB-2410 - от 32 до 255, некоторые модели позволяют определить лишь один символ (но их в расчет принимать далее не будем). Поэтому модуль Scr24 (листинг 1) не является в этом смысле универсальным. Идея проста и заключается в том, что с экрана "снимаются" матрицы пикселов 8x24 (8 вертикальных столбцов по 24 пиксела). Массив таких матриц и переопределяет некоторый диапазон номеров символов. Кодирование принтера происходит следующим образом.

Определяются три группы по 8 байтов, например:

```
(128,0,0,0,0,0,0,0),    - 8 игл, верхняя группа;
( 0,0,0,0,0,0,0,0),     - 8 игл, средняя группа;
( 0,0,0,0,0,0,0,1)      - 8 игл, нижняя группа.
```

Заданный таким образом и загруженный в принтер символ вызовет удар самой верхней и самой нижней иглы. Но символ - это несколько вертикальных колонок, поэтому в принтер необходимо подать чередующиеся тройки байтов (листинг 1, метод THardCopy24.FormChar). Синтаксис команды формирования символа может быть разным для различных принтеров.

Формат приведенной здесь команды соответствует модели "Amstrad LQ 3500 di 24 pin", на которой проходило тестирование Scr24.



РЕДАКЦИЯ ГАЗЕТЫ
"ЧЕЛОВЕК И КОМПЬЮТЕР"
ПРЕДЛАГАЕТ

Альбом принципиальных схем персональных компьютеров IBM PC XT/AT/SuperAT-386 и их периферийных устройств.
Включает в себя: две различные схемы материнской платы IBM PC XT; восемь схем материнской платы IBM PC AT; три SuperAT-386; в том числе и собранных на сверхбольших чипах; от двух до четырех различных схем следующих устройств: адаптеров мониторов, контроллеров коммуникационных портов ввода-вывода, контроллеров HARD и FLOPPY дисков, аналоговые платы HARD и FLOPPY дисков, видеомониторов, клавиатур, манипуляторов "мышь", принтеров и блоков питания (всего 218 страниц формата А3). **Цена 8990 руб.**

Книга "Элементная база IBM PC AT (от стандарта INTEL до сверхбольших чипов)".
На низком аппаратном уровне рассматриваются распространенные комплекты микросхем, применяемых при производстве системных плат компьютеров семейства IBM PC AT. На уровне внутренних блок-схем рассматриваются принципы функционирования, включение в схему. Подробно рассказывается о назначении и типах всех сигналов. Приведены назначения всех внутренних регистров микросхем, способы их программирования и инициализации в различных режимах работ. **Цена 3990 руб.**

Книга "Техническое описание видеоадаптеров и видеомониторов EGA, VGA, SuperVGA".
Описаны структурные и принципиальные схемы адаптеров и мониторов, приведено полное описание регистров EGA и VGA, цоколевки БИС ряда распространенных комплектов фирм "CHIPS & Technologies", "UMC" и др. Рассматриваются назначение основных сигналов и временные диаграммы. **Цена 2890 руб.**

Книга "Техническое описание и ремонт блоков питания IBM PC XT/AT/SuperAT-386 и их периферийных устройств".
Рассмотрены принципы построения блоков питания персональных компьютеров и их периферийных устройств. Приведено полное описание отечественной ИС М1114ЕУ3 (аналог микросхем TL494c, mPC494c, IR3M02, MB3759), предназначенной для управления однотактными и двухтактными импульсными стабилизирующими преобразователями напряжения. Подробно описаны принципиальные электрические схемы блоков питания. Изложены алгоритмы поиска неисправностей и методы их устранения. Представлены спецификации используемых элементов и их аналоги. **Цена 2850 руб.**

Накопители на дисках в IBM PC XT/AT и их контроллеры. Цена 2990 руб.

Справочное пособие по сверхбольшим чипам. Цена 390 руб.

Пособие по инсталляции и использованию ОС NOVELL 2.15. Цена 2590 руб.

Программа STARTEST с описанием методики ремонта портов ввода/вывода. Цена 1290 руб.
Программа SCREENTEST с описанием методики ремонта мониторов. Цена 1190 руб.
Пакет программ COM_KEYBORD с описанием методики ремонта периферийных устройств компьютеров IBM PC XT/AT/SuperAT-386. Цена 890 руб.

Все цены даны с учетом НДС. При покупке за наличные предоставляется значительная скидка.
(095) 275-24-28, 275-43-36.

Модуль Scr24 позволяет получить только копию всего экрана, поскольку из-за относительно небольших (~ 80x60 мм при HExplode=2) размеров получаемой картинки печать участка не представляет интереса.

Для тестирования необходимо запустить программу Test (листинг 2). ■

Листинг 1

```
Unit Scr24;

{*************************************************}
{                                                 }
{  Demo unit                                      }
{  by Nik.N.Kosarev                1990           }
{                                  ph.439-37-33   }
{*************************************************}

interface

uses
   Graph,
   Printer,
   Screen; { см. предыдущий номер "Монитора" ! }
type

   THardCopy24 = object(THardCopy)
        MinNChar,
        MaxNChar:byte;
        HExplode:byte;
        ColArray: array [0..2,0..1304] of byte;
        constructor Init(InitHExplode:byte);
        procedure FormChar(CoordLeft,
                           CoordRight:integer;
                           NumberChar:byte);
        procedure PrintLine(CoordLeft,CoordRight:Integer);
        procedure SetPrinter(StepPrn:byte);
        procedure Printing;
   end;

implementation

constructor THardCopy24.Init(InitHExplode:byte);
begin
   X1:= 0;
   Y1:= 0;
   X2:= 640;
   Y2:= GetMaxY;
   MinNChar:= 31;                  { Переопределяются номера с 31-го}
   MaxNChar:= 110;                 { по 110-й. }
   HExplode:= InitHExplode;        { Коэффициент горизонтального расширения,}
                                   { диапазон 1..3, при HExplode = 2 на VGA }
                                   { искажение отсутствует. }
end; { Init }

{ Процедура формирования символа }
procedure THardCopy24.FormChar(CoordLeft,
                               CoordRight:integer;
                               NumberChar:byte);
var
   i,k: integer;
   WidthChar:byte;
begin
   WidthChar:= (CoordRight-CoordLeft+1)*HExplode;

   write(lst, #27,'&'+chr(0)+chr(NumberChar)+  { Формирование символа.}
                      chr(NumberChar)+  { Номер загружаемого символа,}
                      chr(0) +          { левый отступ от соседнего символа,}
                      chr(WidthChar) +  { ширина символа в колонках,}
                      chr(0));          { правый отступ от соседнего символа.}
   { тройки байтов }
   for i:= CoordLeft to CoordRight do
        for k:= 1 to HExplode do
             write(lst,chr(ColArray[0,i]),
                  chr(ColArray[1,i]),
                  chr(ColArray[2,i]));
end; { FormChar }

procedure THardCopy24.SetPrinter(StepPrn:byte);
begin
   write(lst, #27#51+chr(StepPrn) );
end; { SetPrinter }

procedure THardCopy24.PrintLine(CoordLeft,CoordRight:Integer);
var i:integer;
begin
   for i:= MinNChar to MaxNChar do begin
        FormChar(CoordLeft,CoordRight,i);
        inc(CoordLeft,8);
        inc(CoordRight,8);
   end; {for}

   write(lst,#27,'%'+chr(1));          { Активизация нового шрифта.}
   for i:= MinNChar to MaxNChar do
        write(lst,chr(i));                    { Печать строки символов.}

   write(lst,#13,#10);                 { Возврат печатающей головки.}
end; { PrintLine }

procedure THardCopy24.Printing;
   var i: integer;
   CoordLeft,
   CoordRight:integer;
begin
   LimUp:= 0;

   repeat
        LimDn:= LimUp+7;
        for i:= X1 to X2 do ColArray[0,i]:= ScanLine(i);
             inc(LimUp,8);
             inc(LimDn,8);
        for i:= X1 to X2 do ColArray[1,i]:= ScanLine(i);
             inc(LimUp,8);
             inc(LimDn,8);
        for i:= X1 to X2 do ColArray[2,i]:= ScanLine(i);
             inc(LimUp,8);

        CoordLeft:= X1;
        CoordRight:= X1+7;

        SetPrinter(24);
        PrintLine(CoordLeft,CoordRight);

        write(lst,#27,'%'+chr(0));   { Деактивизация нового шрифта.}

   until LimUp>Y2+24;

end;{ Printing }

end.
```

■

Листинг 2

```
program Test24;
{*************************************************}
{                                                 }
{  Test program                                   }
{                                                 }
{*************************************************}

uses Scr24,
   Graph,
   printer;
var
   GraphDriver,
   GraphMode    : Integer;
   HardCopy     : THardCopy24;

procedure StartGraph;
begin
   GraphDriver := Detect;
   InitGraph(GraphDriver,GraphMode,''); { не забудьте указать путь }
   if GraphResult <> grOk then begin          { к драйверу EgaVga.Bgi   }
        writeln('Graphics error');
        Halt(1);
   end; { if }
end;

begin
   StartGraph;
   Rectangle(0,0,640,GetMaxY);
   HardCopy.Init(2);
   HardCopy.Printing;
   CloseGraph;
end.
```

■

МИКРОИНФОРМ

NOVELL

Локальные вычислительные сети

- Файловые серверы на базе IBM PC AT-386, IBM PC AT-486
- Сетевые жесткие диски от 300 Mb и выше
- Сетевые адаптеры и оборудование 3COM
- Платы дискового сопроцессора
- Коаксиальные и оптико-волоконные кабели
- Интеллектуальные устройства бесперебойного элетропитания

Программное обеспечение

- Операционные системы NetWare v2.2, v3.11
- Замена ОС NetWare на более поздние версии
- Средства работы с базами данных SQL, XQL, Btrieve
- Компилятор С для ОС NetWare
- Программы мониторинга и диагностики сети
- Программы сопряжения ОС NetWare с ОС UNIX

Авторизованный Учебный Центр фирмы

NOVELL

при СП МИКРОИНФОРМ

приглашает на курсы по изучению локальных вычислительных сетей

Инструкторы фирмы NOVELL проводят обучение
NetWare 386 v3.11 : System Manager Service & Support.

По окончании курса выдается сертификат фирмы NOVELL

Слушатели обеспечиваются учебными пособиями фирмы NOVELL на русском языке.

Ежегодно проводятся экзамены на степень Certified NetWare Engineer.

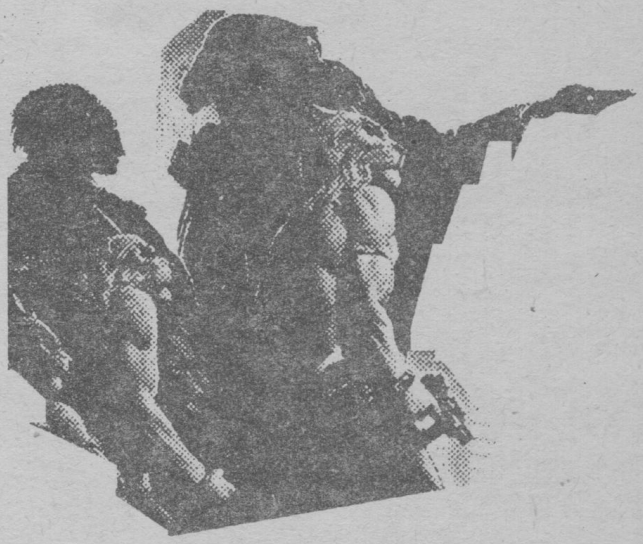
Мы ждем Вас!

СП МИКРОИНФОРМ 113184 Москва ул. Островского д. 44 Коммерческий отдел
Учебный центр

Телефон : (095) 233 0006
(095) 235 7700

Телефакс: (095) 235 1053
Эл. почта: LEXICON@MICRO.MSK.SU.

Adventure Games: эволюция жанра



Игры этого жанра - совершенно особый удивительный мир фантастических и романтических сюжетов, необычных персонажей, тонкого юмора и неожиданных коллизий. У нас их часто называют "текстовыми" играми, хотя они уже давно перестали быть чисто текстовыми. Им скорее подошел бы термин "диалоговые игры". Понятно, что любая игра представляет собой диалог с компьютером, но если в играх других направлений диалог является необходимым элементом системы управления игрой, то здесь диалог в значительной мере является самоцелью программы, и успех играющего в первую очередь зависит именно от того, насколько он сумеет установить общий язык общения с машиной.

"ADVENTURE" переводится на русский язык как "ПРИКЛЮЧЕНИЕ", но этот перевод совсем не адекватен тому, что на самом деле имеется в виду, и не может быть принят. По всей видимости, наиболее целесообразно пока именовать этот класс программ неформальным термином "адвентюры", определения для которого пока нет, но описание которого может быть дано.

Проводившиеся за рубежом социологические исследования среди групп пользователей ПЭВМ однозначно показывают, что если в первые месяцы общения с компьютером наиболее популярными являются игры жанров ARCADY и ACTION, то пользователи со стажем в 2-3 года отдают безусловное предпочтение стратегическим играм и жанру ADVENTURE.

В нашей стране, к сожалению, этот жанр получил развитие пока только среди ограниченных групп энтузиастов и здесь можно выделить две основные причины:

- английский язык не является родным;
- недостаточная информированность играющих в компьютерные игры о приемах и методах работы с адвентюрами, отсутствие практических навыков.

К счастью, обе эти проблемы не являются неразрешимыми. Во-первых, если недостаточное знание английского языка может быть препятствием для работы, то, наоборот, игра может быть полезной школой английского языка. Что же касается второй проблемы, то она тоже поддается решению. Как показывает исследование, основных приемов и методов не так уж и много, они вполне поддаются классификации и детальному разбору.

Какие же причины толкают нас на то, чтобы выделить именно этот класс программ из моря игрового программного обеспечения? Их несколько. Во-первых, это обучающий эффект в плане английского языка. Во-вторых, умение наладить диалог с ранее не известной пользователю программой - неотъемлемый элемент общей компьютерной грамотности. Можно с большой уверенностью утверждать, что тот, кто способен разобраться со сложной адвентюрой, способен освоить и новую для себя прикладную систему или язык программирования. (А нашим программистам в силу известных причин нередко приходится делать и то и другое при отсутствии полноценной фирменной документации - факт скорее печальный, чем отрадный).

Значительный вес в адвентюрах играет и общекультурный аспект. Они часто строятся по мотивам древних легенд, баллад, по литературным или кинематографическим произведениям. Из ранних программ стоит отметить игры "Tir-Na-Nog" и "Dun Darach" фирмы GARGOYLE GAMES, созданные по мотивам древних кельтских и шотландских сказаний. Из современных наиболее интересными представляются программы "Camelot" и "Conquest of the Longbow" фирмы SIERRA ON-LINE, написанные по мотивам легенд о короле Артуре и рыцарях Круглого Стола, а также на основе баллад о Робине Гуде. Нередко эти игры можно рассматривать как некоторую разновидность интерактивного литературно-художественного произведения, в котором играющий может в определенной (хотя и ограниченной) мере влиять на развитие сюжета и на поведение персонажей. Часто уровень такого произведения достаточно высок и представляет самостоятельную художественную ценность.

Развитие жанра привело к одному ранее неожиданному эффекту. Оказалось, что адвентюрные программы, в отличие от прочих игр, не только не вызывают психологического отрыва "хекера" от социальной среды, а наоборот способствуют установлению новых неформальных связей, расширению круга интересных знакомств и, в конечном итоге, помогают некоторым неумеренным "игрокам" скорректировать свое поведение в работе с компьютером. В странах Западной Европы отмечено образование целых социальных групп (в первую очередь людей одиноких, пенсионеров, отставников и инвалидов), для которых контакт с единомышленниками на почве общих интересов придал дополнительный положительный окрас повседневному быту.

Успешное прохождение адвентюрной игры как правило требует от играющего нестандартного мышления и особой логики в принятии решений. Применяемые при этом методы относятся к эвристическим: методы аналогий, исключения, инверсии,- и многие другие. С этой точки зрения игру можно считать целенаправленным исследованием. Принятые в игре решения помогают формированию особого подхода к творческим проблемам, который всегда будет полезен в научных исследованиях, программировании, изобретательстве, в сфере образования и в других областях научной и гуманитарной деятельности.

Первые шаги

Адвентюрные игры явились едва ли не самыми первыми компьютерными программами наряду с простейшими позиционными играми типа "Крестиков-ноликов".

Представьте себе средний вычислительный центр конца 70-х годов. Огромные блоки дорогостоящей аппаратуры работают круглосуточно (аппаратура дорогая и не может простаивать). Персонал несет утомительные ночные дежурства. Именно в таких условиях и появились первые адвентюрные программы. Сначала они создавались полуподпольно, по секрету от руководства, но постепенно приобрели известность. Первая коммерческая программа появилась под названием "ADVENTURE", и это название предопределило наименование всего направления. Впоследствии этот термин еще не раз эксплуатировался в ранних программах - "Dungeon Adventure", "Adventure Quest", "Colossal Adventure", "Classic Adventure" и многие другие.

Поскольку о графических дисплеях в то время речь еще не шла, первые адвентюры разрабатывались как чисто текстовые диалоговые игры. Система управления программами довольно проста. Пользователю сообщается описание обстановки, в которой он находится, далее ему предлагается ввести команду для выполнения одного из нескольких возможных действий (GO NORTH, GO EAST..., TAKE SWORD), после чего выдается описание новой ситуации, если команда прошла, или ответная реакция, если команда не может быть исполнена. Основным содержанием игры стало перемещение в игровом пространстве и постепенное его исследование - элемент, сохранившийся в качестве одного из основных и до наших дней.

Такая система управления предопределила и принципы построения структурной схемы программ, которая включала в себя КАРТУ ЛОКАЦИЙ (массив параметров, определяющих игровое поле), СЛОВАРЬ КОМАНД, СЛОВАРЬ ОБЪЕКТОВ, ТАБЛИЦУ ОПИСАНИЙ, ТАБЛИЦУ РЕАКЦИЙ и ФЛАГОВЫЙ МАССИВ, содержащий информацию о координатах игрока и его параметрах). Так, например, взятие ключа (TAKE KEY) в одной локации может включать флаг, позволяющий открыть дверь (OPEN DOOR) в другой локации.

Основным приемом прохождения таких программ стало исследование словаря и подбор команд для выполнения поставленной задачи. В зависимости от класса программы это требовало логического мышления, интуиции или простого перебора методом проб и ошибок.

Первые программы имели весьма ограниченное количество локаций и слабо развитые словари объектов и команд, но, тем не менее, они сумели получить признание любителей за счет одного малоисследованного психологического приема. Дело в том, что адвентюрная программа как бы соткана из логических и лексических головоломок, разгадка которых требует глубокого проникновения в ситуацию, вживания в нее. Таким образом, при ограниченных возможностях художественного выражения авторской идеи (по сравнению с кино, телевидением, театром, литературой и музыкой) достигается ощутимый эффект присутствия.

Смежные жанры

Конечно, первые адвентюрные программы развивались не в изоляции от других направлений компьютерных и не только компьютерных игр.

Практически одновременно с первыми адвентюрами появились игры типа D&D (DUNGEONS AND DRAGONS). Генеральная цель этих программ - пройти игровое пространство (как правило, это лабиринт) и выполнить конечную задачу, например, спасти своих друзей, как в программе "Nightmare on Elm Street", выпущенной фирмой MONARCH SOFTWARE по одноименному фильму ужасов "Кошмары на улице Вязов" ("Кошмары Фредди"). При этом на всем протяжении маршрута приходится сражаться с "темными силами" - скелетами, привидениями, оборотнями и т.п.). Техника управления игрой типа D&D не требует развитого диалога с машиной, а победная стратегия заключается в том, чтобы отыскать маршрут, следуя которому можно побеждать противников не в произвольном порядке, а так, чтобы их сила возрастала постепенно и, соответственно, постепенно нарастали бы сила, боевой опыт и оснащенность Вашего героя.

У этого жанра уже была сложившаяся предыстория в виде настольных игр, но он, тем не менее, так и не завоевал особой популярности у игроков со стажем вследствие утомительного однообразия сюжетной ли-

нии и слабого эффекта "реального присутствия". И нам не стоило бы и говорить об этом направлении, если бы в определенный период не произошло взаимообогащения игр D&D и ADVENTURE. Стратегия выбора маршрута с учетом накопленного боевого опыта и приобретенного вооружения вошла в адвентюрные игры и стала элементом многих из них. Одна из талантливейших фирм - LEVEL NINE в своем ставшем классическим шедевре "Price of Magic" использовала этот элемент в нескольких локациях. Иногда фирмы делают это с целью затруднения доступа в наиболее важные пункты игрового поля, а иногда для того, чтобы стимулировать малоподвижного и пассивного героя к более активным действиям.

Даже в высокоразвитых современных программах, в той или иной степени задействующих боевое противоборство персонажей, вводится сюжетная линия, следуя которой герой может пройти боевое обучение в серии схваток и подготовиться к ответственным действиям в более сложных ситуациях. Фирмы иногда проявляют чудеса выдумки для того, чтобы сделать схватку как можно более интеллектуальной и отойти от традиций аркадных игр, в которых победа в схватке достигается интенсивным стучанием по клавиатуре. Так, например, в программе "The Secret of Monkey Island" фирмы LUCASFILM GAMES схватка на шпагах имитируется оживленным диалогом. Вовремя вставленная остроумная фраза эквивалентна хорошо поставленному удару, а достойный ответ на словесный выпад противника - удачному парированию укола. Но надо еще найти себе учителя, а также спарринг-партнеров. Также мало что удастся сделать игроку в программе "Hero Quest II" фирмы SIERRA ON-LINE, если он не найдет себе учителя фехтования и не возьмет у него несколько уроков. В этом случае его действия за пределами городских стен всегда будут сопровождаться риском для жизни.

Еще одним жанром, оказавшим влияние на развитие адвентюрных программ, явились так называемые игры RPG (ROLE-PLAYING-GAMES). Они тоже сформировались до появления первых компьютеров и развивались сначала как "игры по переписке".

Игры были командными. Фирма-организатор разрабатывала сюжет и рассылала правила всем желающим включиться в игру за скромную абонементную плату. Из участников формировались команды по 4-6 человек в каждой. В соответствии с пожеланиями игроков им расписывались роли (Воин, Разведчик, Маг, Колдунья, Священник и т.п.). Каждому участнику сообщалось описание обстановки в той локации, в которой он находится, и дальше он действовал по своему усмотрению, перемещаясь в игровом пространстве, манипулируя объектами в ходе игры и общаясь с другими персонажами. Игрок заполнял свои команды на стандартных отрывных бланках, отправлял их по почте и получал в ответ новое описание ситуации. Поскольку нормально поставленная почтовая система позволяет в течение суток обменяться письмами до двух раз в пределах одного городка, игра протекала достаточно динамично, но, тем не менее, растягивалась на месяцы.

Самое основное, что позаимствовали адвентюрные игры у RPG, - кооперативное решение задач. Под управлением играющего находится не один персонаж, а несколько, причем с различными характеристиками. Это немного усложняет систему управления, но вносит известное разнообразие в поиски оптимальной стратегии. Пользователь получает несколько дополнительных степеней свободы, которые связаны с возможностью перераспределять имеющиеся у него объекты между персонажами и использовать героев в соответствии с их игровыми амплуа. По-видимому, одним из первых образцов программного обеспечения этого направления стала программа "Enigma Force" фирмы BEYOND SOFTWARE, в основу которой положен сюжет, сходный с популярным научно-фантастическим фильмом "Чужие" ("Aliens"). Из программ последних лет у нас широко известны "Eye of Beholder I" и "Eye of Beholder II" компании WESTWOOD ASSOCIATES. Они имеют довольно развитую систему манипуляции объектами.

Последним пограничным направлением, на котором хотелось бы остановиться, является упомянутый выше жанр интерактивного литературного произведения. Первым развитым образцом этого направления, по-видимому, следует считать юмористический сериал "The Secret Diary of Adrian Mole" фирмы MOSAIC. Программа представляет дневник среднего английского мальчишки-сорванца, постоянно попадающего в нелепые истории. На каждом шаге пользователю предлагается выбрать один из возможных вариантов разрешения текущей проблемы, что приводит к еще более смешным и забавным коллизиям.

В настоящее время хорошая адвентюрная программа, как правило, имеет художественную ценность и, пусть в малой степени, но может считаться литературным произведением. Так, например, трудно сказать, чем является программа "Colonel's Bequest" фирмы SIERRA ON-LINE, - адвентюрной игрой или интерактивной пьесой в стиле А.Кристи.

Дальнейшее развитие ранних образцов адвентюрных игр шло по пути введения в программы элементов искусственного интеллекта (ИИ). Первым коммерческим образцом такой программы считается "Valhalla", выпущенная фирмой LEGEND в начале 1983 года по мотивам скандинавских мифов. Концепция ИИ в этих играх основывалась на том, что главному герою, а также и побочным персонажам, с которыми он может встретиться на игровом поле, приписываются некоторые свойственные человеку атрибуты - сила, интеллект, очарование, магические способности, боевой опыт и т.п. По ходу игры эти атрибуты могут меняться в зависимости от того, какой характер принимают действия главного героя. На основе этих первичных атрибутов с помощью системы эвристических уравнений рассчитываются более сложные комплексные параметры, которые определяют поведение персонажей в игре. Например,

МУЖЕСТВО = СИЛА * БОЕВОЙ ОПЫТ / ИНТЕЛЛЕКТ + ОТВЕТСТВЕННОСТЬ

Такое уравнение, например, будет означать, что малосимпатичные и безответственные персонажи будут идти на конфликт с Вами, если они абсолютно уверены в своих силах либо очень глупы.

Один и тот же персонаж может проявить в отношении главного героя дружелюбие или враждебность. Первичный контакт может закончиться диалогом и взаимопомощью при наличии "интеллекта" и "очарования", а может привести к боевой схватке. Да и сама схватка носит заранее непредсказуемый характер, так как в зависимости от фактора "мужества" противник может стоять до конца под грузом атрибута "ответственности", а может и убежать. Убегая, он может бросить оружие, а может и нет, в зависимости от того, какое значение имеет фактор "силы" в момент бегства. Нужно еще учесть, что атрибуты могут меняться не только во времени, но и в координатах. Так, дракон, которому положено охранять сокровища, имеет более высокий атрибут "ответственности" в тех локациях, в которых эти сокровища есть.

Вы можете с интересом провести время, исследовав, как меняются атрибуты персонажей по мере развития событий в программах "Eye of Beholder" (эти атрибуты можно получить для каждого героя нажатием клавиши "I").

Введение элементов ИИ в адвентюрные программы позволило сделать их более гибкими и разнообразить сюжет без значительного увеличения объема памяти, что было особенно важно для бытовых 8-разрядных машин. Вам не удастся дважды совершенно одинаково сыграть в одну игру. К сожалению, в связи с быстрым прогрессом средств вычислительной техники насущная необходимость в этом отпала и сейчас трудно встретить адвентюру, реализующую концепцию ИИ в большом объеме, хотя отдельные элементы иногда встречаются.

С другой стороны, прогресс технических средств и, в первую очередь, средств графической поддержки предопределил дальнейшие направления эволюции жанра. Она продолжалась в развитии системы управления и системы представления информации, о чем мы поговорим в следующей статье цикла, посвященного этому жанру, а пока специально для любителей адвентюрных игр "ИНФОРКОМ" объявляет новый конкурс. Десять победителей, правильно назвавших наибольшее количество программ фирмы SIERRA ON-LINE, получат памятные призы. Наиболее полный ответ будет опубликован. В качестве подтверждения того, что Вы реально видели эти программы, Вам надо для каждой программы указать имя пускового EXE модуля, его длину и дату создания. Срок подачи ответов - до 1 марта 1993 года.

Напоминаем наш адрес: 121019 Москва, Г-19, а/я 16, "ИНФОРКОМ".

Желаем успеха! ■

КОМПЛЕКС ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ

QUICKTOOLS FOR PARADOX

версия 1.0

Если Вы разрабатываете прикладные системы на базе СУБД Paradox 3.5, то QuickTools 1.0 for Paradox - инструмент для Вас ! То, на что у Вас раньше уходили недели, теперь Вы сможете сделать за минуты и часы.

По своим возможностям QuickTools не уступает, а кое в чем и превосходит Paradox 4.0 WorkShop. При этом в отличии от Paradox 4.0 WorkShop QuickTools под управлением Paradox 3.5 с успехом работает на PC/AT с 1 Мб RAM.

QuickTools - это комплекс инструментальных средств, которые позволяют автоматизировать разработку интерфейса прикладных систем.

ТОО 'ИНТЕГРИРОВАННЫЕ КОМПЬЮТЕРНЫЕ СИСТЕМЫ'

Разработчики приложений могут использовать QuickTools для моделирования сложных систем, уменьшая объем кодирования на языке PAL и снижая тем самым трудоемкость их разработки.

Используя QuickTools1.0 Вы можете.

- Создавать горизонтальные, вертикальные и матричные меню.
- Связывать с элементами меню действия, активизируемые при их выборе.
- Формировать заставки (TitleScreens).
- Создавать экраны Help и связывать их с элементами меню.
- Получать программы на языке PAL, осуществляющие вывод заставок, меню, подсказок, и активизировать действия, связанные с выбором в меню.
- Использовать библиотеку типовых процедур на языке PAL.

PARADOX, PAL, WorkShop - торговые марки или зарегистрированные торговые марки Borland International, Inc.

За справками обращаться : тел. 238-99-33; факс 238-32-78; адрес 117419, Москва а/я 765.

Почему я лютый враг компьютерных игр

Г. Шабат

1. Они создают особый тип психического отклонения ребенка. "Заболевший" компьютерными играми ребенок распознается по лихорадочному блеску в глазах, когда он колотит по клавишам, иногда по патологической компьютерно-игровой эрудиции, иногда по повышенному вниманию к связанным с компьютерами второстепенным деталям и всегда - по готовности жертвовать едой и сном ради компьютерных игр. Ответственно заявляю, что такой ребенок потерян для математики, подозреваю, что то же касается других высших форм интеллектуальной деятельности.

2. Они поглощают массу компьютерного времени. В нашей стране принято жаловаться на острую нехватку компьютеров, действительно, научный работник или студент, который захочет что-нибудь посчитать, поработать над текстом, попробовать программу машинного перевода или напечатать график, столкнется с массой проблем. Однако в огромном количестве школ стоят компьютерные классы; зайдите наугад в любую: если класс не заперт, Вы, вероятно, увидите людей разных возрастов, остервенело гоняющих чудовищ по экранам (ну, или делающих что-то более изощренное; автор вынужден признаться в глубоком невежестве в компьютерных играх как таковых). Ту же картину Вы найдете в разнообразнейших учреждениях: на немногих имеющихся компьютерах играют дети сотрудников, или (совестно писать, но ведь правда же!) сами сотрудники.

3. Они вносят вклад в создание мира муляжей, угрожающего развитой цивилизации. Смею думать, что играть в футбол здоровей, чем в Тетрис, что подросток, раздобывший запретную игру, где в награду за правильно введенное английское слово на экране раздевается красотка, и проведший за этой игрой часы, утрачивает что-то, чего ему будет недоставать при прикосновении к живой девушке.

Ответим на стандартные аргументы апологетов компьютерных игр.

Они приучают к клавиатуре. Ну уж во всяком случае не лучше, чем созидательная деятельность на компьютере (к тому же многие игры приучают лишь к виртуозному владению стрелками и джой иком). Научите ребенка писать интересные программы, дайте в руки редактор, с помощью которого он может красиво оформлять свои мысли (только не забудьте освободить время от компьютерных игр, чтобы эти мысли появлялись), и через пару месяцев ребенок будет обращаться с клавиатурой с уверенностью профессионального оператора.

Они обучают английскому. Во-первых, мягко выражаясь, не шекспировскому, во-вторых, заведомо неактивному.

Ребенку хочется и нравится играть. Ответьте сами на аналогичный аргумент относительно алкоголя и наркотиков, и вывод будет очевиден: не держите вблизи ребенка компьютерных игр.

Они развивают быстроту реакции. Во-первых, не лучше пинг-понга, во-вторых, крайне односторонюю.

Они демонстрируют программные продукты высокого класса. Этот аргумент серьезнее предыдущих, и автор согласен, что у многих начинающих программистов стремление представить продукт в выгодном свете сформировалось под влиянием наиболее профессионально написанных компьютерных игр (автор считает также несомненным, что юноша, побывавший в Нью-Йоркском баре, узнает о культуре поведения больше, чем тот, который бывал только в московской пивной). Беда, однако, в том, что фирмы и частные программисты, разрабатывающие компьютерные игры, озабочены прибылью, а не просвещением, так что секреты их мастерства остаются скрытыми от заинтересованного подростка. В результате некоторые молодые люди тратят массу бесплодного времени на проникновение в эти тайны, азартно состязаясь друг с другом в рассекречивании кодов и других премудростях (представьте себе, что было бы, если бы мы учили наших студентов интегрированию, демонстрируя блестящее мастерство, но скрывая основные приемы, поскольку последние составляли бы коммерческую тайну!). Другие пытаются написать игры сами, пользуясь доступными им скромными средствами, и, как правило, не достигают больших успехов - автор судит по собственным многолетним наблюдениям. Мне известно о некоторых исключениях: победители соответствующих конкурсов демонстрируют довольно милые результаты. В любом случае идеалом является возможность продажи. По моему скромному мнению, этот навязанный взрослыми идеал отвлекает детей от более высоких целей.

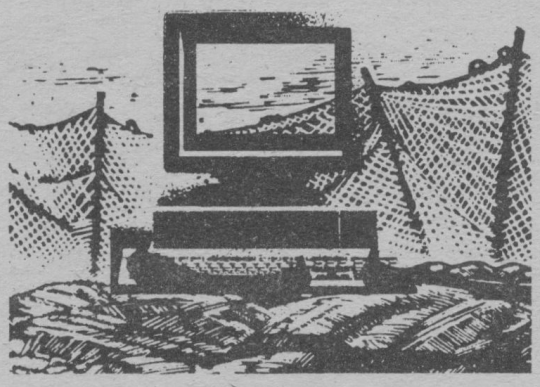
Закончим некоторым более общим соображением. Мы присутствуем при явлении, которое в гиперболизированной форме было описано в мрачной утопии Хаксли "Бравый Новый Мир". В этой утопии описано стандартизированное общество, в котором одним из условий стабильности является интенсивное производство специализированного оборудования для электронных спортивных игр. Потребность в продуктах этого производства обеспечивается навязыванием обществу (на подсознательном уровне) соответствующего морального стандарта: уклонение от проведения существенной части дня за электронными играми считается безнравственным и карается.

Мне, конечно, возразят, что человек, не тратящий время на компьютерные игры, ничем пока не рискует. Это верно, но позвольте все-таки зафиксировать определенный сдвиг в современном общественном сознании: **многим из нас внушили, что компьютерные игры цивилизованнее традиционных.**

Доказательство: случалось ли Вам видеть, чтобы в приемной декана факультета престижного университета студенты забивали "козла" или резались в подкидого (тоже, кстати, занятия, развивающие смекалку и психологию и при этом не грозящие вирусами нужным для работы устройствам). Думаю, нет. А чтобы в компьютерные игры? Мне лично - увы.

Доказательство закончено. Автор готов ответить на возражения. ■

Что такое IPX и как его использовать в своих целях



Часть 2

А. Ермолов

Резидентные программы, работающие с IPX

В прошлый раз мы написали программу, которая передает по сети простейшее сообщение от одной станции к другой. Программа, особенно на принимающей станции,получилась чрезвычайно неудобной. После ее запуска машина переходила в состояние долгого и нудного ожидания, и ни одна полезная программа на ней не выполнялась. Чтобы избежать подобных неприятных эффектов, сетевые программы, как правило, оформляют в виде резидентных программ (TSR). Написание таких программ само по себе дело довольно трудное. Если Вы хотите подробнее ознакомиться с ним, могу порекомендовать статью "Этот безумный, безумный, безумный мир резидентных программ" в журнале "Компьютер пресс" (номера 4, 5 за 1992-й год).

Я же в этой статье опишу только особенности использования такими программами возможностей IPX. Поэтому программа, которую мы в этот раз напишем, будет весьма несовершенной: в ней нет средств защиты от повторной загрузки, она не может выгружаться из памяти без перезагрузки системы (этим, впрочем страдают и многие коммерческие TSR), и, наконец, - верх падения для системного программиста - она написана на Си, а не на ассемблере. Последнее, впрочем, может быть даже интересно: Вы увидите, как можно вполне корректно оформлять на Си самые низкоуровневые программы.

Программа, которую мы напишем в данном разделе статьи, будет делать то же самое, что делает программа приема, приведенная в прошлом номере журнала. То есть она принимает и выдает на дисплей строку - сообщение, передаваемое с другой станции. Правда, я не хочу связываться с такими проблемами, как нереентерабельность MS-DOS, и поэтому принятую строку буду выдавать в предпоследнюю строку дисплея при помощи прямой записи в видеопамять адаптера (предполагается, что он работает в символьном режиме).

Основной принцип написания резидентных программ, работающих с IPX, заключается в применении пользовательских программ, вызываемых IPX при регистрации какого-либо события. Такие программы в документации Novell называются ESR (Event Service Routine). Для того, чтобы сообщить IPX, какую программу надо запускать, адрес входа ESR надо записать в соответствующее поле управляющей структуры (ECB). Структура ECB вместе с рядом других полезных сведений была приведена в прошлом номере журнала "Монитор". На ее повторение, к сожалению, места нет.

ESR не является программой обработки прерываний, в частности, ее разработчик не обязан заботиться о сохранности всех регистров. Однако надо учитывать, что эта программа вызывается из программы обработки аппаратного прерывания IPX. Поэтому желательно, чтобы ESR срабатывала быстро и работала при закрытых прерываниях (кстати, при ее запуске прерывания закрыты). Так что если необходимо по событию IPX осуществить запись на диск, ввод с клавиатуры или еще какое-либо действие, требующее длительного времени или открытых прерываний, то лучше, чтобы программа ESR только выставляла признак, который должен периодически проверяться программой, осуществляющей основную работу. Как это сделать, описано в упомянутой статье.

При вызове ESR ей передаются в качестве параметра адрес ECB (в регистровой паре Es:Si) и признак вида управляющего блока в регистре Al. Что обозначает последний параметр, объясняется ниже. Пока рассматриваются программы приема сообщений, нам этот параметр не нужен. Передача адреса ECB удобна в том случае, когда одна программа обрабатывает несколько сообщений, принимаемых через разные сокеты. Использование нескольких ECB полезно также при организации очереди сообщений. Именно таким образом используется эта возможность в приведенном примере. Конечно, маловероятно, что кто-то сумеет так быстро послать сообщения с двух машин, что возникнет очередь (мне это не удалось), но попытаться можно.

В прошлый раз при трансляции программ использовалась модель Large. На этот раз программа приема написана для модели Small. Поэтому, кстати, текст Ipx.h (листинг 3). несколько изменился. Программа ввода в новом варианте (листинг 1) состоит из двух частей: инициализации, которая получает управление при старте, и программы (ESR) обработки события - прихода сообщения.

Если используется модель, отличная от Large, надо учитывать, что ESR должна использовать длинный возврат. Поэтому она объявлена с атрибутом far. С тем же атрибутом объявлены и все структуры IPX. Последнее сделано потому, что подпрограмма при запуске не восстанавливает сегментные регистры, а оставляет там случайные числа. При попытке адресовать переменную, которая определена вне ее тела, ESR использует случайное значение сегментного регистра данных (Ds) со всеми вытекающими отсюда последствиями.

Программа инициализации посылает IPX запрос на ввод сетевого сообщения и оставляет модуль в памяти в качестве резидентной программы. Обратите внимание на то, что для вычисления необходимого программе объема памяти в первых строках запоминается начальное состояние регистров стека (Ss:Sp). Для этого используется фрагмент, написанный на встроенном ассемблере. Начальное значение пары Ss:Sp определяет верхнюю границу памяти, занимаемой модулем. Значение нижней границы определяется положением PSP (program segment prefix). Таким образом объем необходимой памяти вычисляется "без затей", то есть с избытком. Например, в памяти остается никому не нужный PSP и сама программа инициализации.

Программа обработки события вызывается IPX с параметрами, передача которых описана выше. В начале программа определяет поле ECB, которое управляло полученным сообщением. Для установления адреса ECB используется тот же прием, что и при определении положения стека программой инициализации. Зная ECB, определяем адрес буфера ввода и распечатываем принятую информацию. Наконец, возобновляется запрос на ввод сообщения, управляемый тем ECB, адрес которого был определен в первых строках программы. В отличие от текста, приведенного в прошлой статье, я написал все обращения к IPX на встроенном ассемблере, раз уж избежать его применения не удалось.

Кроме указанных действий, программа ввода указывает цветом строки номер ECB, ответственного за прием сообщения. Это позволит Вам изучить логику работы IPX с ECB. Можете поэкспериментировать с различными версиями IPX: NetWare 2.1x, NetWare 2.2, NetWare 3.xx, NovellLite ... Всегда ли получаются одинаковые результаты?

Кроме двух описанных частей, в состав программы приема входит еще подпрограмма выдачи строки в видеопамять дисплея. Ее структура, по-моему, ясна из текста. Поясню для тех, кто не знает, что буфер видеопамяти в символьном режиме обычно начинается с адреса 0xb800. В буфере по четным адресам располагаются выводимые символы, а по нечетным - байты, определяющие цвет символа и фона.

Что может быть событием IPX?

В предыдущем разделе я упоминал о событии, управляемом IPX. Из контекста, по-моему, ясно, что приход сообщения может быть событием IPX, а также то, что IPX может обрабатывать и другие события. Правда, таких событий немного: кроме уже указанного приема сообщения, это завершение передачи сообщения (завершение обозначает только, что сообщение успешно отправлено, но не то, что оно принято) и истечение заданного временного интервала (в двух вариантах).

Обработка передачи сообщения при помощи ESR возможна, но не очень полезна, так как передача пакета максимального размера происходит достаточно быстро. Лучше использовать зацикленное ожидание (с анализом поля InUseFlag). Для защиты от зацикливания можно поставить счетчик циклов ожидания. По моим наблюдениям, даже для сравнительно медленных плат ArcNet и на машине 33 МГц количество циклов ожидания при нормальной работе сети не должно превышать 0x8000. При превышении счетчиком циклов этого значения на более или менее обычной технике можно смело кричать "Караул !".

Пример подпрограммы передачи пакета с зацикленным ожиданием и с контролем продолжительности этого ожидания приведен в листинге 2. В данном примере используется новая функция IPX - снять событие, управляемое указанным ECB (Cancel). Программа передачи сообщения возвращает код завершения. Он равен коду завершения, формируемому IPX, если события удалось дождаться, или 0xFC (этот код завершения соответствует принудительному завершению процедуры при помощи вызова Cancel), если не удалось. Когда Вы будете пробовать приведенную программу передачи, у Вас могут возникнуть проблемы с имитацией ошибки передачи (если только Ваши аппаратные средства не отечественного производства). Для получения качественной и надежной ошибки передачи в сети ArcNet достаточно отключить кабель от адаптера связи. То же для Token Ring. Но в сети Ethernet ничего столь же простого и дешевого посоветовать не могу.

Третий вид события - это истечение интервала времени. Использование этого вида событий обсуждается в следующем разделе.

Служба времени IPX

Службу времени программа IPX поддерживает в двух вариантах: через протокол IPX и через упрощенный протокол -

"Вскрытие" парольной защиты в операционных системах DR DOS

Ключи и замки - это лишь для честных людей, а вот от ... защиты нет. Однако бывают случаи, когда Вы сами потеряли ключ и не можете попасть в свою же собственную квартиру. Как быть в этом случае? Ломать дверь - жалко, бить окно (а если на дворе зима?) - еще хуже.

Так же бывает и при работе с компьютером: поставлен пароль, а потом забыт. Что же делать? Переформатировать "винчестер" или подбирать ключ для снятия пароля? Лично я выбираю второе.

Со мной произошел такой случай: после отработки архиватора lha версии 2.12 с командой m и ключами -r1x1 образовался архив объемом более 2 Мбайтов, к которому не было доступа. Работа компьютера проходила под управлением операционной системы DR DOS 5.0. Складывалось впечатление, что после упаковки архива и переименования его из временного в итоговый система сама установила атрибуты: hidden и пароль. Сложилась такая ситуация, что архив есть, а воспользоваться им невозможно, так как пароль неизвестен. В операционных системах DR DOS 3.41 и 5.0 файлы и каталоги паролируются с помощью утилиты Password или из оболочки ViewMAX. Здесь же ни первой, ни второй в системе не было, а пароль появился. В чем дело?..

Выяснилось, что дело в операционной системе и программе lha. Паролирование архива происходит в том случае, если в переменной окружения указан каталог, в действительности отсутствующий в системе, например:

SET TMP=C:\TEMP ,

а на диске C отсутствует или стерт каталог TEMP. Для исключения таких случаев я рекомендую добавить в autoexec.bat следующую строку:

IF NOT "%TEMP%"=="" MD %TEMP% > NUL.

Теперь о снятии защиты в DR DOS. Для этого необходимо воспользоваться утилитой DiskEdit из набора Norton Integrator или аналогичной из PCTools. Необходимо найти на диске место, где в таблице размещения файлов записана информация по интересующему Вас файлу. В следующем за названием файла байте (смещение 0Bh) записаны атрибуты этого файла (для каталога - аналогично файлу), затем следуют четыре байта, два из которых (смещение 0h) используются для установки пароля (рисунок). Следует заменить эти два байта на 00H, снять атрибут hidden (установить значение 20h), записать внесенные изменения на диск, и проблема решена.

```
 0 1 2 3 4 5 6 7  8 9 A B C D E F

48 49 44 4F 53 20 20 20 - 53 59 53 20 00 00 00 00 HIDOS   SYS....
00 00 00 00 00 00 00 30 - 17 17 51 00  82 48 00 00 .......0..Q..H..
4B 45 59 42 20 20 20 20 - 43 4F 4D 22 00 00 0F 9C KEYB   COM"....
00 00 00 00 55 05 00 30 - 17 17 0F 00 1F 87 00 00 ....U..0........
4D 45 4D 20 20 20 20 20 - 45 58 45 20 00 00 00 00 MEM     EXE....
00 00 00 00 00 00 00 30 - 17 17 49 00 2D 2C 00 00 .......0..I.-,..
4D 45 4D 4D 41 58 20 20 - 45 58 45 20 00 00 00 00 MEMMAX  EXE....
00 00 00 00 00 00 00 30 - 17 17 4C 00 A6 08 00 00 .......0..L.....
```

Рис. Фрагмент таблицы размещения файлов

Ю. Крючков ■

Asynchronous Event Sheduller (AES). Эти протоколы отличаются форматами ECB. Протокол IPX использует обычный формат управляющего блока, а протокол AES - сокращенный (специальный) формат (листинг 3). При вызове ESR вид ECB, управлявшего свершившимся событием, передается в качестве параметра в регистре AL. Если использовался протокол IPX, Al содержит 0x00, в противном случае - 0xFF. Номер функции, передаваемый IPX в регистре Ax, равен 5 при использовании полного формата ECB (протокол IPX) и 7 при использовании специального формата.

Протокол IPX полезно использовать для отсчета временных интервалов в тех случаях, когда почему-либо нужно, чтобы для оповещения об истечении тайм-аута и для оповещения о приеме сообщения использовался один и тот же ECB. В остальных случаях лучше использовать протокол AES. Пользуясь AES, Вы можете не так беспокоиться о быстродействии своей подпрограммы. Она может работать достаточно долго, в том числе и при открытых прерываниях. Самое худшее, к чему это может привести, - это к задержке на импульс таймера сообщения о другом временном интервале. Требования к ESR, вызываемой из IPX, жестче: надо опасаться как потери прерывания (если работа ведется при закрытых прерываниях), так и путаницы из-за повторного вызова (если ESR выполняет команду STI, то есть разрешает прерывания).

Пример использования службы времени дан в листинге 4. Это та же программа приема, которая описана в первом разделе этой статьи, но если в течение полминуты сообщение не поступило, она оповещает сидящего за пультом об этом прискорбном факте. Программа помещена не полностью. То, что совпадает с программой 1, опущено.

Обратите внимание, что для отсчета временного интервала используется специальный формат ECB. В первых строках программа наряду с адресом ECB запоминает состояние регистра Al, которое впоследствии используется для того, чтобы узнать, что случилось: пришло сообщение или же истек тайм-аут. Если пришло сообщение, то ожидание тайм-аута снимается, а в конце программы восстанавливается, чтобы полминуты отсчитывались от момента прихода последнего сообщения.

Итак, мы увидели, как пользоваться регистрацией временных событий - "будильником", входящим в состав службы времени программы IPX. Но эта служба содержит и обыкновенные "часы", на которые Ваша программа может "посмотреть" когда захочет. Это делается при помощи вызова GetIntervalMarker. Точнее сказать, вызов GetIntervalMarker - это обращение не к часам, а к "секундомеру": он возвращает не астрономическое время, а двухбайтовое количество импульсов таймера с неизвестного момента. Тем не менее, такое средство полезно для резидентных программ, потому что позволяет обходить нереентерабельность MS-DOS. Код функции - 8, значение метки времени возвращается в регистре Ax.

Прочие возможности IPX

В заключение статьи я совсем кратко расскажу о тех возможностях IPX, о которых не было упомянуто ранее (полный список вызовов см. в справочном приложении к статье).

Прежде всего об уже упомянутых вызовах: посылка сообщения возможна в сокеты не только удаленной, но и вызывающей машины. В этом Вы можете убедиться, запустив на Вашей машине резидентную программу приема и послав ей с помощью программы передачи сообщение. При посылке сообщения себе диагностируется ситуация, когда сообщение посылается в сокет, в котором сообщения никто не ждет.

Следующий вызов - GetInternetworkAddress (номер функции - 9) - позволяет программе определить номер локального участка сети и адрес узла, на котором она исполняется. Вызов GetInternetworkAddress записывает в десятибайтовый буфер сперва 4 байта номера локального участка, а затем шестибайтовый сетевой адрес. Пример использования этого вызова дан в листинге 5.

Функция с номером 2 (Get Local Target) используется для доступа к узлам, находящимся на удаленных локальных участках сети. Она по заданному полному адресу узла (номер локальной сети плюс сетевой адрес узла) определяет шестибай-

Справочное приложение

ПЕРЕЧЕНЬ ФУНКЦИЙ IPX

Открыть сокет (Open Socket)

Номер функции - 0x00.
Аргументы: Al - флаг продолжительности жизни, Dx - номер сокета, 0 - динамически выделяемый.
Возвращаемые параметры: Al - код завершения, Dx - номер открытого сокета.

Закрыть сокет (Close Socket)

Номер функции - 0x01.
Аргументы: Dx - номер сокета.

Получить непосредственный адрес (Get Local Target)

Номер функции - 0x02.
Аргументы: Es:Si - указатель на 12-байтовый буфер полного адреса узла, Es:Di - указатель на 6-байтовый буфер приема адреса.
Возвращаемые параметры: Al - код завершения, Cx - транспортная задержка передачи пакета.

Послать пакет (Send Packet)

Номер функции - 0x03.
Аргументы: Es:Si - указатель на ECB.

Инициировать прием пакета (Listen Packet)

Номер функции - 0x04.
Аргументы: Es:Si - указатель на ECB.

Задать временное событие в протоколе IPX (Shedule Event)

Номер функции - 0x05.
Аргументы: Ax - величина временного интервала в импульсах таймера, Es:Si - указатель на ECB полного формата.

Снять запрос (Cancel Event)

Номер функции - 0x06.
Аргументы: Es:Si - указатель на ECB.
Возвращаемый параметр: Ax - код завершения.

Задать временное событие в протоколе AES (Shedule Special Event)

Номер функции - 0x07.
Аргументы: Ax - величина временного интервала в импульсах таймера, Es:Si - указатель на ECB полного формата.

Получить значение интервального таймера (GetIntervalMarker)

Номер функции - 0x08.
Возвращаемый параметр: Ax - значение таймера.

Получить свой сетевой адрес (GetInternetworkAddress)

Номер функции - 0x09.
Аргументы: Es:Si - указатель на 12-байтовый буфер приема адреса.

Передать управление (Relinquish Control)

Номер функции - 0x0A.

Разорвать связь (Disconnect From Target)

Номер функции - 0x0B.

товое значение, которое надо поставить в поле ImmediateAddress ECB, чтобы послать сообщение. Если интересующий Вас абонент находится на том же локальном участке, что и Вы, функция возвратит тот же сетевой адрес, что был указан в запросе, в противном случае - сетевой адрес моста.

Данная функция используется редко, так как обычно никто не знает ни полного, ни какого-либо иного адреса узлов сети. Для определения адресов абонентов обычно поступают другим способом: посылают инициализационное сообщение всем узлам (адрес broadcast - FF FF FF FF FF FF) и затем собирают отклики всех возможных абонентов. При этом в ECB приема автоматически заносится то самое значение ImmediateAddress, которое возвратила бы функция Get LocalTarget.

Наконец, две последние функции - Relinquish Control (0x0A) и Disconnect From Target (0x0B) - и вовсе экзотические.

Первая используется, в основном, в многозадачных системах для того чтобы освободить процессор. Правда, она может понадобиться, если Ваше приложение работает на невыделенном сервере или мосте (в этом случае DOS имитируется средствами многозадачной ОС).

Вторая функция нужна только в тех сетях, которые физически поддерживают транспортный уровень. Она позволяет разорвать соединение, установленное аппаратно. ■

Журнал "Монитор" приглашает к сотрудничеству:

- Дилеров по распространению журнала
- Специалистов по анализу состояния компьютерного рынка
- Авторов оригинальных статей по программированию и вычислительной технике

Тел. (095)238-99-33
Факс (095)238-32-78
Адрес для писем: 117419 Москва, а/я 765.

Размышления об отличиях в расчетах дисперсии и среднекватратичного отклонения в SuperCalc

Применение интегрированного пакета или электронных таблиц, совмещенных с функциями деловой графики, часто оправдано при экспрессной обработке результатов измерений. Наличие встроенных статистических функций позволяет очень быстро оценить результаты того или другого эксперимента, представить полученные данные в графическом виде, определить погрешность результатов. Однако применение встроенных функций может иногда оказаться и медвежьей услугой, так как методика расчетов этих величин может сильно отличаться от правил, принятых по ГОСТ 11.004-74 (Прикладная статистика. Правила определения оценок и доверительных границ для параметров нормального распределения). Рассмотрим это на примере статистических функций STD и VAR, входящих в пакет программ SuperCalc.

Функции VAR и STD предназначены для вычисления дисперсии и среднеквадратичного отклонения (СКО) величин, заданных в списке аргументов. Как известно, дисперсия характеризует степень отклонения отдельных величин из заданного списка от их среднего значения, а среднеквадратичное отклонение вычисляется как квадратный корень из дисперсии. Но есть два варианта расчета дисперсии и связанного с ней среднеквадратичного отклонения. В случае если генеральное среднее нормального распределения известно (a), при расчете дисперсии берут сумму квадратов отклонений отдельных наблюдений от генерального среднего, деленную на число наблюдений (1). В противном случае среднее находится как сумма квадратов отклонений отдельных наблюдений от среднего (3), деленная на число наблюдений без единицы (2). Таким образом, встает вопрос о том, по какой из этих формул производится расчет дисперсии и среднеквадратичного отклонения функциями VAR и STD пакета SuperCalc. Ведь если число наблюдений мало или, точнее, недостаточно велико, то при делении суммы квадратов отклонений на число наблюдений минус единица может наблюдаться существенное расхождение в результате.

$$СКО_1 = \sqrt{\frac{(x_i - a)^2}{n}} \qquad (1)$$

$$СКО_2 = \sqrt{\frac{(x_i - \bar{x})^2}{n - 1}} \qquad (2)$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad (3)$$

$$K = \sqrt{\frac{n}{n - 1}} \qquad (4)$$

где n - число наблюдений;
xi - отдельное наблюдение из выборки;
a - генеральное среднее;
x - математическое ожидание.

Проведенный анализ функций VAR и STD пакета SuperCalc показывает, что в эти функции заложено предположение о том, что генеральное среднее нормального распределения известно, и для корректного применения этих функций с учетом указанного ГОСТа их необходимо домножить на коэффициент k (4), в том случае, если генеральное среднее неизвестно, а определяется по (3). Значение дисперсии тогда можно определить в SuperCalc, занеся в клетку формулу:

VAR(список)*COUNT(список) / (COUNT(список) - 1),

а среднеквадратичное отклонение

STD(список) * SQRT(COUNT(список) /(COUNT(список) - 1)).

Вполне логично сделать вывод о том, что такое же разночтение может наблюдаться и в других интегрированных пакетах и электронных таблицах и, прежде чем применять аналогичные функции, следует выполнить контрольные расчеты для уточнения примененной в конкретном случае формулы расчета.

Ю.Крючков ■

Листинг 1. Резидентная программа приема сообщения

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <e:\article\progs\ipx.h>

#define TargetSocket 0x08
#define Screen 0xb800
#define StrLen 0x50
#pragma check_stack (off)
#pragma check_pointer (off)
#pragma intrinsic ( _enable, _disable )

/* Уменьшим стек */
  extern unsigned _stklen = 512;

/* Конец и начало модуля */
  char _huge *tsrtop;
  char _huge *tsrbottom;
  /**/
/*
  Программа приема и распечатки строки символов
  (резидентный вариант)
*/

/* Переменные общего пользования */
  unsigned char _far RecieveString [80];        // Принимаемая строка
  IPXHeader _far Header;
  ECB_2 _far My_ECB;

/* Выдача строки в видеоамять */
void out_str (char _far *OutBuf, unsigned char Collor, short Str,
short Col) {

/*
  Параметры: выдаваемая строка,
                цвет,
                строка начала выдачи,
                столбец начала выдачи
*/
  unsigned int ScrOff;        // Смещение в буфере экрана
  unsigned int i;             // Счетчик символа в строке
  unsigned char _far *CountPtr;      //Адрес текущего символа
  ScrOff = ( StrLen*Str + Col ) * 2;
  i = 0;
  FP_OFF (CountPtr) = ScrOff;
  FP_SEG (CountPtr) = Screen;
  while (OutBuf [i] != 0x00) {
        CountPtr [i*2] = OutBuf [i];
        CountPtr [i*2 + 1] = Collor;
        i++;
  }
}
void _far My_ESR () {   //Резидентная программа приема
  ECB_2 _far *My_ECB_ptr;
  union REGS inregs, outregs;
  struct SREGS segreg;
  /* Распечатка строки в предпоследнюю строку экрана */
  out_str (RecieveString, 0x2f, 22, 0);

/* Переинициализация приема сообщения */
  My_ECB_ptr = &My_ECB;

  /* Заполнение ECB */
  _asm      NOP // Так надо по причине ошибки в трансляторе
  My_ECB_ptr->SocketNumber = TargetSocket;
  My_ECB_ptr->FragmentCount = 2;
  My_ECB_ptr->FragmentAddress_1 = &Header;
  My_ECB_ptr->FragmentSize_1 = header_length;
  My_ECB_ptr->FragmentAddress_2 = &RecieveString;
  My_ECB_ptr->FragmentSize_2 = sizeof RecieveString;
  My_ECB_ptr->ESRAddress = (void _far *)(&My_ESR);

  /* Прием сообщения. Операторы Си для тех, кто работает с турбо
  inregs.x.bx = IPX_listen;
  inregs.x.si = FP_OFF ( My_ECB_ptr );
  segreg.es = FP_SEG ( My_ECB_ptr );
  int86x (IPX_interrupt, &inregs, &outregs, &segreg);*/

  _asm {
  MOV Ax, SEG My_ECB
  MOV Es,Ax
  MOV Si, OFFSET My_ECB
  MOV Bx, IPX_listen
  INT IPX_interrupt
  }
}
void main () {
  ECB_2 _far *My_ECB_ptr;
  unsigned tsrsize;  // Размер модуля
  union REGS inregs, outregs;
  struct SREGS segreg;

  /* Вычисление размера модуля */
  _asm {
  MOV WORD PTR tsrtop [0], sp
  MOV WORD PTR tsrtop [2], ss
  };
        FP_SEG (tsrbottom) = _psp;
        FP_OFF (tsrbottom) = 0;
        tsrsize = ((tsrtop - tsrbottom) > > 4) + 1;

  /* Заполнение ECB */
  My_ECB.ESRAddress = (void _far *)My_ESR;
  My_ECB.SocketNumber = TargetSocket;
  My_ECB.FragmentCount = 2;
  My_ECB.FragmentAddress_1 = &Header;
  My_ECB.FragmentSize_1 = header_length;
  My_ECB.FragmentAddress_2 = &RecieveString;
  My_ECB.FragmentSize_2 = sizeof RecieveString;

  /* Открытие сокета */
  inregs.x.bx = IPX_open;
  inregs.h.al = 0xFF;
  inregs.x.dx = TargetSocket;
  int86 (IPX_interrupt, &inregs, &outregs);

  /* Выдача кода завершения */
  if (outregs.h.al != 0x00) {
        printf ("Ошибка открытия сокета.  Код завершения %x \n",
                outregs.h.al); exit (1); }

  /* Прием сообщения */
  inregs.x.bx = IPX_listen;
  My_ECB_ptr = &My_ECB;
  inregs.x.si = FP_OFF ( My_ECB_ptr );
  segreg.es = FP_SEG ( My_ECB_ptr );
  int86x (IPX_interrupt, &inregs, &outregs, &segreg);

/* Остаемся резидентом */
  printf ("Программа приема установлена\n");
  _dos_keep (0, tsrsize);
  }
```

Листинг 2. Процедура передачи сообщения

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <ipx.h>


/*
  Подпрограмма посылки сообщения со сбросом после 0x8000
  безрезультатных циклов проверки завершения передачи

  Параметр: ECB
  Возвращает код завершения
*/
#define TRUE 0xFF
#define FALSE 0x00
#define TA 0x8000

int sendPacket (ECB_2 My_ECB) {
  int i; // Счетчик циклов ожидания
  unsigned char CancelSgn;   // Признак зависания передачи
  union REGS inregs, outregs;
  struct SREGS segreg;

/* Начальные присвоения */
  i = 0;
  CancelSgn = TRUE;

  /* Запрос на вывод */
  inregs.x.bx = IPX_send;
  inregs.x.si = FP_OFF (My_ECB);
  segreg.es = FP_SEG (My_ECB);
  int86x (IPX_interrupt, &inregs, &outregs, &segreg);

  /* Ожидание ввода */
  while ((i < TA) && (My_ECB.InUseFlag != 0x00)) {
        i++;
        if (My_ECB.InUseFlag == 0x00)        // Передача произошла
        CancelSgn = FALSE;
  }

  /* Если передача не произошла, снимаем запрос к IPX */
  if (CancelSgn) {
        inregs.x.bx = IPX_cancel;
        inregs.x.si = FP_OFF (My_ECB);
        segreg.es = FP_SEG (My_ECB);
        int86x (IPX_interrupt, &inregs, &outregs, &segreg);
  }

  /* Передача кода завершения */
  return My_ECB.CompletionCode;
}
```

Листинг 3. Константы и структуры IPX

```
/*
  Константы и структуры IPX
*/

#define IPX_interrupt 0x7A     // Программное прерывание IPX
#define header_length 30 // Длина заголовка пакета

/* Функции IPX */
#define IPX_open 0x00
#define IPX_close 0x01
#define IPX_localTarget 0x02
#define IPX_send 0x03
#define IPX_listen 0x04
#define IPX_event 0x05
#define IPX_cancel 0x06
```

```
#define IPX_specialEvent 0x07
#define IPX_interval 0x08
#define IPX_GetInternetwork 0x09
#define IPX_control 0x0A
#define IPX_disconnect 0x0B

/* Заголовок пакета */
typedef struct{
  unsigned int Checksum;
  unsigned int Length;
  unsigned char TransportControl;
  unsigned char PacketType;
  unsigned char DestantionNetwork [4];
  unsigned char DestantionNode [6];
  unsigned int DestantionSocket;
  unsigned char SourceNetwork [4];
  unsigned char SourceNode [6];
  unsigned int SourceSocket;
  } IPXHeader;

/* Управляющий блок с одним фрагментом данных */
typedef struct {
  void _far *LinkAddress;
  void _far *ESRAddress;
  unsigned char InUseFlag;
  unsigned char CompletionCode;
  unsigned int SocketNumber;
  unsigned char IPXWorkspace [4];
  unsigned char DriverWorkspace [12];
  unsigned char ImmediateAddress [6];
  unsigned int FragmentCount;
  void _far *FragmentAddress;
  unsigned int FragmentSize;
  } ECB_1;

/* Управляющий блок с двумя фрагментами данных */
typedef struct {
  void _far *LinkAddress;
  void _far *ESRAddress;
  unsigned char InUseFlag;
  unsigned char CompletionCode;
  unsigned int SocketNumber;
  unsigned char IPXWorkspace [4];
  unsigned char DriverWorkspace [12];
  unsigned char ImmediateAddress [6];
  unsigned int FragmentCount;
  void _far *FragmentAddress_1;
  unsigned int FragmentSize_1;
  void _far *FragmentAddress_2;
  unsigned int FragmentSize_2;
  } ECB_2;

/* Управляющий блок службы времени */
typedef struct {
  void _far *LinkAddress;
  void _far *ESRAddress;
  unsigned char InUseFlag;
  unsigned char WorkSpace [5];
  } SpecialECB;

/* Сетевой адрес плюс непосредственный адрес узла */
typedef struct {
  unsigned char NetworkNumber [4];        // Номер локального участка
  unsigned char NodeAddress [6];          // Сетевой адрес
  unsigned int Socket;                    // Номер сокета
  unsigned char ImmediateAddress [6];     // Непосредственный адрес
  } IntAddr;
```

Листинг 4. Использование службы времени

```
. . . . . .

/* Переменные общего пользования */
  unsigned char _far RecieveString_1 [80];    // Буферы ввода
  unsigned char _far RecieveString_2 [80];
  unsigned char _far TimeOutString [20] = "Time Out\0";
  IPXHeader _far Header; // Заголовок принимаемого сообщения
  ECB_2 _far My_ECB_1;        // Два ECB
  ECB_2 _far My_ECB_2;
  SpecialECB _far ECB_TO;     //Для отсчета временного интервала

. . . . . .

void _far My_ESR () {       //Резидентная программа приема
  ECB_2 _far *_My_ECB_ptr;
  unsigned char AESFlag; // Признак сообщения службы времени
  union REGS inregs, outregs;
  struct SREGS segreg;
  char _far * _RecieveString;              // Адрес буфера ввода
  unsigned char CountCollor;               // Цвет строки выдачи

  /* Определение ECB и буфера ввода */
  _asm {
  MOV WORD PTR _My_ECB_ptr[0], SI
  MOV WORD PTR _My_ECB_ptr[2], Es
  MOV BYTE PTR AESFlag, al
  }
  _RecieveString = _My_ECB_ptr -> FragmentAddress_2;

  /* Распечатка в предпоследнюю строку экрана */
  if (AESFlag) {  // Сетевое сообщение
       Обработка принятого сообщения

  /* Отмена отсчета временного интервала */
  _asm {
  MOV Ax, SEG ECB_TO
  MOV Es, Ax
  MOV SI, OFFSET ECB_TO
  MOV Bx, IPX_cancel
  INT IPX_interrupt
  };
  }
  else {              // Обработка сообщения службы времени
       out_str (TimeOutString, Collor_TO, 22, 0);
    /* Заполнение ECB */
    _My_ECB_ptr-> ESRAddress = (void _far *)(&My_ESR);
  }
  /* Запрос службе времени: переустановка временного интервала */
  _asm {
  MOV Ax, WORD PTR _My_ECB_ptr[2]
  MOV Es,Ax
  MOV SI, WORD PTR _My_ECB_ptr[0]
  MOV Ax, TimeOut
  MOV Bx, IPX_specialEvent
  INT IPX_interrupt
  };
}
void main () {

. . . . . .

  /* Заполнение ECB */

. . . . . .

  ECB_TO.ESRAddress = (void _far *)My_ESR;

. . . . . .

  /* Запрос службе времени */
  inregs.x.bx = IPX_specialEvent;
  inregs.x.ax = TimeOut;
  My_ECB_ptr = &ECB_TO;
  inregs.x.si = FP_OFF ( My_ECB_ptr );
  segreg.es = FP_SEG ( My_ECB_ptr );
  int86x (IPX_interrupt, &inregs, &outregs, &segreg);
/* Остаемся резидентом */
  printf ("Программа приема установлена\n");
  _dos_keep (0, tsrsize);
  }
```

Листинг 5. Распечатка сетевого адреса рабочей станции

```
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <ipx.h>
#define IPX_interrupt 0x7A
/*
  Программа выдачи сетевого адреса своей машины
*/
/*
  Шестнадцатеричная распечатка байта с лидирующим нулем
*/
void printHH (unsigned char b) {
  printf ("%1x%1x", b / 0x10, b % 0x10);
  };
void main () {
  struct {   // Сетевой адрес
       unsigned char InterNetWork [4];   // Номер локального участка
       unsigned char NetworkAddress [6];      // Физический сетевой адрес
  } NetAddress;
  unsigned char far *Buf_ptr;          // Адрес принимаемой строки
  union REGS inregs, outregs;
  struct SREGS segreg;
  int i; // Счетчик цикла
  /* Прием номера локального участка сети */
  inregs.x.bx = IPX_GetInternetwork;
  Buf_ptr = &NetAddress;
  inregs.x.si = FP_OFF ( Buf_ptr );
  segreg.es = FP_SEG ( Buf_ptr );
  int86x (IPX_interrupt, &inregs, &outregs, &segreg);
  /* Печать номера локального участка */
  printf ("InterNetWork Address = ");
  for (i = 0; i < sizeof (NetAddress.InterNetWork); i++) {
       printHH (NetAddress.InterNetWork[i]);
  };
  printf ("\n");
  /* Печать сетевого адреса адаптера */
  printf ("NetworkAddress = ");
  for (i = 0; i < sizeof (NetAddress.NetworkAddress); i++) {
       printHH (NetAddress.NetworkAddress[i]);
  };
  printf ("\n");
}
```

Оптимизация

в Zortech C++

А. Сидоров

Эффективность работы программиста и пользователя в первую очередь зависит от эффективности программного обеспечения, которое они используют. Бесспорно, на очень хорошей технике и не слишком эффективная программа будет работать вполне удовлетворительно. Но где эта техника?

Для не избалованных передовой компьютерной техникой отечественных пользователей проблема эффективности программного обеспечения стоит очень остро. А может это и к лучшему? Пытаясь втиснуться в прокрустово ложе имеющихся аппаратных ресурсов, отечественные программисты вынуждены изобретать более эффективные алгоритмы и добиваться высокого уровня оптимизации, перекраивая исходный код по очень жестким критериям.

Производительность ПО может трактоваться по-разному. Можно добиваться более высокой скорости выполнения, не заботясь о размерах. Или, наоборот, втискивать в несколько Кбайтов могучие системы. В общем случае экономия памяти приводит к увеличению времени работы программы, и наоборот.

Главные инструменты любого системного программиста на персональных и мини-ЭВМ - Си и ассемблер. Технология оптимизации на ассемблере достаточно прозрачна и очень зависит от характера конкретных приложений. Рассмотрим оптимизацию на примере компиляторов Си.

Под оптимизацией принято понимать довольно широкий набор методов, повышающих производительность ПО. Методы принято разделять на машинно-зависимые и машинно-независимые. Основными областями машинно-независимой оптимизации обычно являются вынесение общих подвыражений из циклов, вычисление постоянных функций и т.д. Методы оптимизации имеют различную степень сложности и эффективности. Простейшие из методов могут дать эффект только для плохо написанных программ, а лучшие из них приносят выигрыш даже при оптимизации мастерски сделанных. Однако прежде чем приступить к оптимизации, следует учесть все затраты, которые будут необходимы.

Каждый из компиляторов проводит оптимизацию, руководствуясь определенными критериями. Так, компиляторы, предназначенные для обучения и снабженные дополнительными средствами отладки и контроля ошибок, позволяют уменьшать время, затрачиваемое на отладку, за счет увеличения времени компиляции или за счет качества кода программы.(Пример - компиляторы Borland). Оптимизирующие компиляторы работают гораздо медленнее, но создают очень эффективные программы. Однако можно добиться большого успеха и не используя оптимизирующий компилятор, а сосредоточившись на изобретении эффективного алгоритма и написании производительного исходного кода.

Последний способ самый качественный, но и самый дорогой. С ходу написать оптимальную (и работающую!) программу может далеко не ординарный программист. И никакой руководитель не будет попусту заставлять заниматься ручной оптимизацией чужих программ подобных асов, так как это обойдется весьма дорого фирме за счет потери производительности . Обычному программисту нужно довольно долгое время для "вылизывания" программ до оптимального состояния. И дело это муторное и неблагодарное.

Люди очень хорошо знают, что такое накладные расходы и как свести их к минимуму. Но далеко не все программисты представляют, во что обходится каждая операция написанного ими кода. Даже многие системные программисты не всегда могут ответить на вопрос, какой лучше использовать оператор для создания "синего" цикла (то есть крутящегося "до посинения"). Почему-то существует мнение, что быстрее всего работает цикл типа

```
while(1) {
.....
}
```

На самом деле обычно предпочтительнее цикл

```
for(;;) {
.....
}
```

В этом можно убедиться, проанализировав выполнимый код в любом из отладчиков.

Разница всего в одну операцию сравнения, но и это при большом количестве итераций или при организации обработки внешнего сигнала может значить очень много.

А сколько встречается программ, где для определения нечетности целого в цикле используется оператор деления по модулю

```
while ( iFlag % 2)
{
....
},
```

вместо более элегантного и, что гораздо важнее, быстрого способа

```
while ( iFlag & 0x0001)
{
.....
}
```

Отличие весьма существенное: в первом случае вызывается целая подпрограмма деления по модулю, а во втором это реализуется всего двумя командами ассемблера, одна из которых - логическая операция AND. Как известно, логические операторы выполняются за минимальное количество тактов процессора.

Со времен IBM 360/370 широко известна хитрость, вошедшая, кажется, во все учебники программирования на

ассемблере : обмен двух байтов местами без использования третьего с помощью трех операторов XOR:

```
XOR A,B
XOR B,A
XOR A,B
```

Почему-то практически никто не использует этот прием в программировании на Си, хотя в нем есть специальная операция "Исключающее ИЛИ", и применять эту операцию можно практически к любым интегральным типам данных:

```
int a,b;
a ^=b;
b ^=a;
a ^=b;
```

Таких секретов, повышающих эффективность программ, существует великое множество, и любой профессиональный программист имеет свои любимые приемы. Простейший и, на мой взгляд, нагляднейший из них предложен в статье асов программирования на Си из Bell Labs фирмы AT&T [1]. Трудно что-либо возразить авторам Си и UNIX !

Я попробовал применить эту методику для оценки способностей оптимизации доступных и "модных" компиляторов. Пришлось несколько скорректировать исходный текст теста, так как он был предназначен для больших компиляторов UNIX. Кроме того, Borland C++ 2.0 и 3.0 отказались воспринимать тест в виде одного модуля, как в оригинале, из-за переполнения внутренних таблиц компилятора. Так неожиданно проявилась невозможность серьезной препроцессорной обработки исходных текстов в Borland C++.

Впрочем, компилятору Microsoft C/C++ 7.0 тоже показалось тесновато в каких-нибудь 4 Мбайтах ОЗУ 486-го процессора. (Вообще странно видеть, как продукт фирмы Microsoft оказывается несовместимым с MS-DOS 5.0 и требует установки драйверов третьих фирм для более или менее нормального функционирования! В расширенном режиме MS Windows этот монстр-компилятор тоже работает через раз).

Компиляторы Zortech и Watcom тоже показали себя в данной ситуации не лучшим образом: при оптимизации выдавали сообщения о невозможности оптимизации части функций. Поэтому программа была разделена на четыре модуля для удобства обработки (листинги 1,2,3,4).

Кроме того,были внесены изменения в систему вывода для удобства обработки результатов с помощью Quattro Pro и статистического пакета ОЛИМП.

Идея теста сама по себе тривиальна. Замеряем время процессора до выполнения оператора и после. Разница дает нам время выполнения оператора. Для большей точности оператор нужно вставить в достаточно большой цикл и внести поправку на время выполнения самого цикла и количество итераций. В программе это сделано с помощью замера выполнения пустого цикла. Осталось добавить некоторую оценку погрешности, и очень удобный инструмент для экспериментальной проверки качества осуществления оптимизации в компиляторах готов!

Тестирование проводилось на IBM AT с 486 процессором, имеющим тактовую частоту 33 МГц с 4 Мбайтами оперативной памяти и 256 Кбайтами кэш, скорость доступа к винчестеру - 17 нс (табл. 1).

Хотя для Watcom и Zortech есть возможность генерации кода для 386/387 и 486 процессоров, были установлены стандартные переключатели для 80286 и для математического сопроцессора 8087. Установка оптимизации - на скорость выполнения кода в ущерб размеру (листинг 5).

Результаты, мягко говоря, странные (табл. 2). Известные оптимизирующие компиляторы показали не слишком высокие результаты, но тем не менее скорость выполнения целиком всей программы оказалась выше у Zortech и Watcom секунд на 10-15 (табл. 3). Совсем плохо дело с Microsoft. Чем это вызвано, трудно объяснить. Можно предполагать, что это связано со странностями реализации обработки значений таймера в разных компиляторах и методами оптимизации кода.

При анализе этих данных на корреляцию проявилась интересная особенность: оба компилятора Borland C++ коррелировались в одну группу, а Watcom и Zortech в другую. Это тем более странно, так как BC 3.0 имеет довольно неплохие оптимизирующие возможности в отличие от BC 2.0. Microsoft и здесь попал

Таблица 1

Компилятор	Параметры компиляции
Zortech C++ 3.0;	
-f	Использовать команды 8087
-2	Генерировать код для 80286
-o+time	Оптимизировать для скорости выполнения
-ozc3tst	Имя выходного файла
Watcom C 8.0;	
-7	Использовать команды 8087
-2	Генерировать код для 80286
-ot	Оптимизировать для скорости выполнения
-fe=wc8tst	Имя выходного файла
Borland C++ 2.0;	
-f87	Использовать команды 8087
-2	Генерировать код для 80286
-O	Оптимизировать для скорости выполнения
-G	
-ebc2tst	Имя выходного файла
Borland C++ 3.0.	
-f87	Использовать команды 8087
-2	Генерировать код для 80286
-O2	Оптимизировать для скорости выполнения
-G	
-ebc2tst	Имя выходного файла
Microsoft C/C++ 7.0	
/FPi87	Включать команды 8087
/f-	Использовать оптимизирующий компилятор
/G2	Генерировать код для 80286
/Ot	Оптимизировать для скорости выполнения
/Ox	Максимальная оптимизация
/Femsc7tst	Имя выходного файла

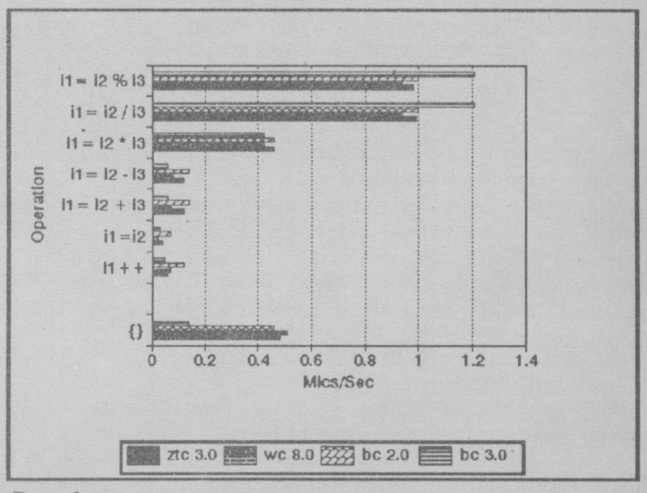


Рис. 1

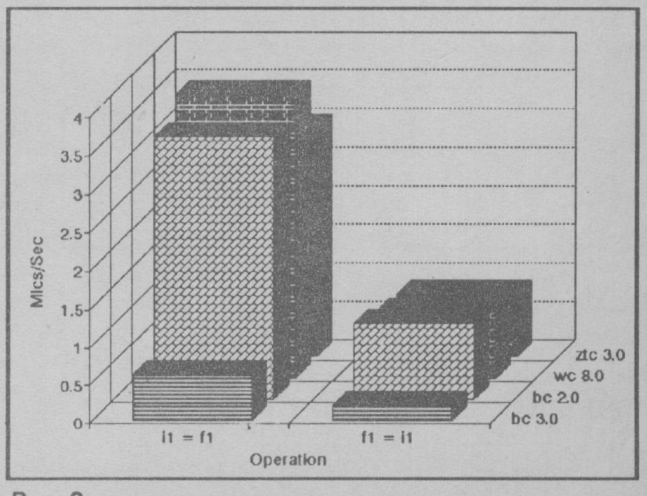


Рис. 2

Таблица 2. Результаты тестирования компиляторов

Операция	ZTC 3.0	WC 8.0	BC 2.0	BC 3.0	MSC 7.0
Пустой цикл (n = 10000000)					
{}	0.48	0.51	0.46	0.14	?
Целые операции (n = 10000000)					
i1++	0.06	0.07	0.12	0.05	?
i1 = i2	0.04	0.02	0.07	0.03	?
i1 = i2 + i3	0.12	0.07	0.14	0.06	?
i1 = i2 - i3	0.12	0.08	0.14	0.06	?
i1 = i2 * i3	0.46	0.42	0.46	0.42	?
i1 = i2 / i3	0.99	0.94	1	1.21	?
i1 = i2 % i3	0.98	0.94	1	1.21	?
Операции с плавающей запятой (n = 10000000)					
f1++	0.99	0.9	0.91	1.6	?
f1 = f2	0.63	0.53	0.58	0.7	?
f1 = f2 + f3	0.97	0.86	0.82	1.18	?
f1 = f2 - f3	1.59	1.48	1.45	1.29	?
f1 = f2 * f3	1.05	0.97	0.92	0.92	?
f1 = f2 / f3	2.93	2.54	3.8	3.54	?
Числовые преобразования (n = 10000000)					
i1 = f1	2.97	3.68	3.43	0.6	?
f1 = i1	0.83	0.86	1	0.19	?
Операции над векторами целых чисел (n = 10000)					
v[i] = i	0.52	0.69	-0.46	0.26	10
v[v[i]] = i	1.72	0.49	0.64	0.06	12
v[v[v[i]]] = i	0.72	0.69	1.74	0.46	10
Управляющие структуры (n = 10000)					
if (i = = 5) i1++	0.52	0.49	0.64	0.06	?
if (i! = 5) i1++	0.72	0.69	0.64	0.06	10
while (i 0) i1++	0.72	0.49	0.46	0.26	12
i1 = sum1(i2)	1.72	1.69	1.74	0.26	22
i1 = sum2(i2,i3)	1.72	1.69	1.74	0.26	22
i1 = sum3(i2,i3,i4)	1.72	1.69	1.74	0.26	22
Ввод / вывод (n = 100000)					
fputs(s,fp)	21.52	21.49	21.52	5.86	252
fgets(s,9,fp)	21.52	21.49	21.52	1.86	154
fprintf(fp,sdn,i)	43.52	65.49	32.51	5.86	548
fscanf(fp,sd,&i1)	31.52	55.49	43.49	7.86	562
Выделение памяти (n = 100000)					
free(malloc(8))	4.8	9.49	10.53	10.86	54
push(i)	9.52	11.49	14.6	14	57538
i1 = pop()	11.52	5.2	11.46	10.86	22
Строковые функции (n = 1000000)					
strcpy(s,s0123456789)	5.78	4.87	6.35	1.14	294.45
i1 = strcmp(s,s)	6.34	6.51	6.46	1.2	366.22
i1 = strcmp(s,sa123456789)	4.46	1.91	4.48	0.86	238.29
Преобразование строки в числа (n = 1000000)					
i1 = atoi(s12345)	6.12	7.29	6.13	1.26	86.8
sscanf(s12345,sd,&i1)	44.52	48.89	29.21	5.46	547.2
sprintf(s,sd,i)	50.12	44.49	28.11	4.86	355.8
f1 = atoi(s123_45)	4.92	6.09	6.13	1.06	71.4
sscanf(s123_45,sf,&f1)	96.12	77.69	40.2	7.66	830.6
sprintf(s,sf62,123.45)	288.52	177.49	61.08	11.46	1170
Математические функции (n = 100000)					
i1 = rand()	6.12	4.89	6.03	5.86	53.8
f1 = log(f2)	28.12	12.49	23.71	24.46	160.4
f1 = exp(f2)	23.72	11.69	29.21	22.45	190
f1 = sin(f2)	24.72	12.69	12.73	22.26	173.6
f1 = sqrt(f2)	3.92	4.89	5.03	4.86	86.8

Таблица 3. Время выполнения теста

Borland C++ 2.0	212.3
Borland C++ 3.0	169.505
Microsoft C++ 7.0	258.37
Watcom C 8.0	141.98
Zortech C++ 3.0	152.2

Рис. 3

в особое положение - данные его тестирования обработать не удалось из-за малого количества достоверных данных.

Конечно, следует учитывать, что этот тест дает ОТНОСИТЕЛЬНЫЕ оценки! Поставив подобный эксперимент и внимательно изучив данные, можно выработать четкую привычку использовать более экономные операторы и функции. Но лучше то же самое пусть проделает компилятор (рис. 1, 2, 3).

Из тестируемых компиляторов Zortech имеет наибольшие возможности для оптимизаций различного типа.

Zortech был пионером среди компиляторов для MS-DOS и OS/2, позволив осуществить глобальную оптимизацию кода. Глобальная оптимизация отличается от локальной тем, что анализирует и оптимизирует код всей функции целиком, а не по частям. Zortech использует так называемую технику анализа потока данных для сбора информации об оптимизируемой функции, что и определяет качество оптимизации.

Технология использования оптимизатора проста: следует отладить нужный модуль в обычном режиме, а затем только откомпилировать с нужными оптимизирующими опциями, так как компиляция с оптимизацией занимает гораздо больше времени, чем обычная. Типичная программа после оптимизации ускоряется от 0 до 30% . Лучше всего использовать оптимизацию в небольших, но часто выполняющихся циклах. Хуже всего поддаются оптимизации программы с большим объемом ввода/вывода и вызова библиотечных функций. Но вместе с тем у Zortech есть редкая возможность эффективно использовать глобальную оптимизацию в программах для MS Windows.

Для включения оптимизации достаточно добавить в строку команд компилятора опцию -o, за которой должны следовать флажки оптимизации, разделенные знаком "+" . Например,

```
ztc program -o-all+space-w
```

В Zortech C++ применяются 12 основных методов оптимизации. Рассмотрим их на конкретных примерах.

1) Constant Propagation (опция +cnp)

До оптимизации:

```
....
double db,vec[N_EL];
....
db = 0.04;
for(i=0; i  db; i++)
    vec[i] = db;
....
```

После оптимизации:

```
.....
double db,vec[N_EL];
.....
db = 0.04
for(i=0; i  0.04; i++)
    vec[i] = 0.04;
....
```

Тут ясно, что переменная db в цикле всегда имеет фиксированное значение 0.04, поэтому и заменяется на константу, что приводит к экономии времени и количества команд.

2) Copy Propagation (опция +cp)

Аналогичен предудущему методу, только вместо константы производится подстановка переменной.

До оптимизации:

```
....
```

После оптимизации:

```
.....
```

```
double db,vec[N_EL],b;          double db,vec[N_EL],b;
....                            ......
db = b;                         db = b;
for(i=0; i  db; i++)            for(i=0; i  b; i++)
    vec[i] = db;                    vec[i] = b;
....                            ....
```

Иногда этот метод может привести к удалению незначащих операторов присваивания (как db = b в данном случае).

3) Dead Assignment Elimination (опция - da)

Удаление операторов присваивания, не имеющих эффекта.

До оптимизации:

```
....
double abs()
{
    double f;
    static double g = 0;
        if (g != 3)
        {
            f = 3;
            g = 3;
        }
}
```

После оптимизации:

```
.....
double abs()
{
    double f;
    static double g = 0;
        if (g != 3)
        {
            g = 3;
        }
}
```

Это экономит время выполнения и размер кода.

4) Dead Code Elimination (опция - +dc)

Исключение пустого ("мертвого") оператора, заведомо никогда не выполняющегося.

До оптимизации:

```
....
const double dmax = 1.0;
if (dmax > 10.0 )
    {
    /* "мертвые" операторы */
    ......
    }
    /* операторы */
    .....
```

После оптимизации:

```
.....
const double dmax = 1.0;
    /* операторы */
    .....
```

5) Dead Variable Elimination, Compute Live Ranges (опция+dv)

Исключение пустых ("мертвых") переменных, объявленных , но никогда не используемых. Такая ситуация возникает довольно часто, сначала после ручной оптимизации кода, а затем после использования предыдущих опций. Иногда свидетельствует о недостаточной проработке алгоритма.

Вторая половина этого метода - вычисление жизненного интервала переменной, то есть тех блоков функции , где эта переменная используется. Часто возникает при неразумном и "щедром" использовании модификатора register с переменными, без учета того, что у процессора число регистров все-таки ограничено.

При использовании этой опции компилятор пытается распределить все эти переменные по времени, если жизненные рамки двух переменных не пересекаются, то для них можно использовать один регистр.

Например:

```
register int i;
    ....
    i = 2; if ( abc ) { /* переменные abc и i могут
использовать один регистр */
    register int a; a = 12;
    }
```

6) Global Common Subexpressions (опция- +gcse)

По возможности вычисляются общие части выражениях для уменьшения вычислений.

7) Loop Invariant Removal (опция - +li)

Вынесение инвариантов циклов за скобки, то есть вынесение выражений, не зависящих от цикла, за его пределы.

До оптимизации:

```
....
double a,b,c;
....
for(i=0; i  b; i++)
    a = func(b*c);
....
```

После оптимизации:

```
.....
double a,b,c,T;
    ......
        T = b*c;
for(i=0; i  b; i++)
    a = func(T);
....
```

8) Loop Induction Variables (опция - +liv)

Замена управления циклом. Это наиболее эффективный метод, повышающий скорость работы.

До оптимизации:

```
....
for (i=0; i; i++)
vec[i*2+3] = 1.2;
```

После оптимизации:

```
.....
double T;
for (T=&vec[3]; T&vec[23]; T += 2)
        *T = 1.2;
```

Чтобы додуматься до такой прелести, надо лет 5 профессионально писать программы на Си и, в основном, транслятоpы! Все преимущества и недостатки Си в этих двух строчках. Очень эффективно, но очень непонятно! Поэтому подобные изыски лучше предоставлять оптимизирующему компилятору.

9) Iterate Until No More Optimizations (опция- +loop)

Продолжать оптимизацию до тех пор пока это возможно, то есть улучшать уже оптимизированный код.

Это очень полезная опция, так как в результате применения предыдущих методов могут возникнуть участки программы , нуждающиеся в применении тех же методов оптимизации. Например, из-за появления пустых операторов, не задействованных теперь переменных, и так далее.

10) Do Our Own Register Assignments (опция- +reg)

Вообще говоря, лучше не использовать модификаторы register в оптимизируемых программах, так как компилятор сам пытается разместить как можно больше переменных в регистрах. Но иногда это необходимо , поэтому эта опция позволяет заставить оптимизатор "уважать" Ваше волевое решение о распределении регистров для переменных. Но тут уж не обижайтесь, если вдруг результат получится хуже !

11) Very Busy Expressions (опция - +vbe)

Оптимизация использования популярных выражений в программах приводит к экономии размеров кода, но не времени.

До оптимизации:

```
....
if (is_true())
    a = b*c;
else
    d = b*c;
```

После оптимизации:

```
    .....
T = b*c;
if (is_true())
    a = T;
else
    d = T;
```

12) Ну и, разумеется, у Zortech C++ есть опции для общего анализа и оптимизации кода на размер (опция - +space) и скорость (опция - +time). Эти опции взаимоисключающие и не могут использоваться совместно.

Кроме того, существует всеобщая опция оптимизации - +all, при помощи которой можно достичь баланса между скоростью и размерами кода, сделав его лучше.

Однако если у Вас в распоряжении техника, намного превосходящая средний уровень, и есть уверенность, что программа будет использоваться только на подобном "железе", то, может быть, быстрее и дешевле вообще не заниматься этим.

Самым же лучшим рецептом для получения качественного кода является один: учиться, учиться и учиться! Только экспериментальный опыт помогает достичь успеха. И если есть возможность перепоручить оптимизацию программы компилятору, лучше это сделать. Ведь главный лозунг программистов:

"Машина должна работать, а человек думать!" ■

Литература

1. John Bentley,Brian Kernighan,Chris Van Wyk. An Elementary C Cost Model// Unix Review. - 1991. - - Vol.9, N 2.
2. Zortech C++ 3.0 Compiler Guide. - Symantec Corporation.

Листинг 1

```
/*
 *    mintime.h
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys\types.h>
#include <time.h>
#include <assert.h>

#define MAXLONG 0x7FFFFFFFL
#define MAXINT    MAXLONG
#undef CLOCKS_PER_SEC
#define CLOCKS_PER_SEC (clock_t)100


long jobclicks(void);
long pop (void);
void push (long l);
void TestMalloc();
void TestControl();
void TestIntVector();
void TestNumConversion();
void TestFloat();
void TestInt();
void TestInOut();
void TestString();
void TestStringConversion();
void TestMath();


#define loop1(CODE) loop1ctr++; \
                    for(i=0;i < n;i++) { CODE; } \
                    loop1next = jobclicks(); \
                    thisclicks = loop1next - loop1start; \
                    sumclicks += thisclicks; \
                    printf("%5d,",thisclicks); \
                    loop1start = loop1next;


#define loop(CODE) printf("\"  %-30s\",",#CODE) ; \
        minclicks = 0; maxclicks = MAXLONG; sumclicks = 0; \
        loop1ctr = 0; \
        loop1start = jobclicks(); \
        loop1(CODE) \
        loop1(CODE) \
        i0 = i1 + i2 +i3; \
        loop1(CODE) \
        i0 = i1 + i2 + i3 - i1 -i2 -i3; \
        loop1(CODE) \
        i0 = i1 + i2 + i3 + i1*i2 -i2*i3 + i1*i3; \
        loop1(CODE) \
        quesstr = ''; \
        if ((float)loop1ctr*(maxclicks-minclicks) > warn*sumclicks ) \
            quesstr = '?' ;\
        lastmics = sumclicks * basefit/(CLOCKS_PER_SEC*n*loop1ctr); \
        printf("%10.2f, %c,\n",(lastmics - basemics),quesstr);


#define title(TEXT) printf("\"%s (n = %ld)\"\n",TEXT,n)
```

Листинг 2

```
/*
 *    Файл MINTIME1.C
 */

#include "mintime.h"

    long loop1start,loop1next;
    float lastmics,basemics;
    long minclicks,maxclicks,sumclicks,thisclicks,loop1ctr;
    long i,n,basen;
    int         i0,i1,i2,i3,i4;
    float f0,f1,f2,f3;
    int         *v;
    char quesstr;
    float basefit = 10000;
    float warn = 0.4;

long jobclicks(void)
    {
        clock_t t;

        t = clock();
        return (t);
    } /* jobclicks */

int   sum1(int a) {return a;}
int   sum2(int a,int b) { return a + b;}
int   sum3(int a,int b,int c) {return a + b+ c;}

void TestInt()
    {
        title("Int Opetations");
        i1 = i2 = i3 = 5;
        loop(i1++)
        loop(i1 = i2)
        loop(i1 = i2 + i3)
        loop(i1 = i2 - i3)
        loop(i1 = i2 * i3)
        loop(i1 = i2 / i3)
        loop(i1 = i2 % i3)
    } /* TestInt */

void TestFloat()
    {
        title("Float Operations");
        f1 = f2 = f3 = 5.0;
        loop(f1++);
        loop(f1 = f2);
        loop(f1 = f2 + f3);
        loop(f1 = f2 - f3);
        loop(f1 = f2 * f3);
        loop(f1 = f2 / f3);
    } /* TestFloat */

void TestNumConversion()
{
    title("Numeric Conversions");
    f1 = 123456.789 ;
    i1 = 12345;
    loop(i1 = f1);
    loop(f1 = i1);
} /* TestNumConversion() */

void TestIntVector()
    {
        n = 10000;
        title("Integer vector Operations");
        v = (int *)malloc(n*sizeof(int));
        for (i=0;i < n;i++)
                v[i] = 0;
        loop(v[i] = i);
        loop(v[v[i]] = i);
        loop(v[v[v[i]]] = i)
        free(v);
    } /*  TestIntVector */

void TestControl()
    {
        title("Control Structures");
        i1 = i2 = i3 = 5;
        loop( if (i = = 5) i1++)
        loop( if (i! = 5) i1++ )
        loop( while (i< 0) i1++ )
        loop( i1 = sum1(i2))
        loop( i1 = sum2(i2,i3))
        loop( i1 = sum3(i2,i3,i4))
    } /* TestControl */

void main(int argc,char **argv)
```

```
    {
        long start,end;
        start = clock();
        if ( argc > 1 && argc < = 3 )
        {
                basefit = atof(argv[1]);
                warn    = atof(argv[2]);
        }
        setbuf(stdout,(char *)NULL);
        printf("\n  Operation                    Clicks for each trial "
                "  Mics/N\n");
        basen = basefit;
        n = basen;
        title("Null Loop");
        i0 = i1 = i2 = i3 = 5;
        f0 = f1 = f2 = f3 = 5.0;

        /* Вычисляем скорость выполнения пустого цикла */
        basemics = 0.0;
        loop({})
        basemics = lastmics;

        TestInt();
        TestFloat();
        TestNumConversion();
        TestIntVector();
        TestControl();
        TestInOut();
        TestMalloc();
        TestString();
        TestStringConversion();
        TestMath();

        end = clock();
        printf("\nВыполнение заняло всего %f сек.",(float)(end-
start)/CLK_TCK);
        return;
    } /* main */
```

Листинг 3

```
/*
 *    Файл MINTIME2.C
 */

#include <string.h>
#include "mintime.h"

extern    long loop1start,loop1next;
extern    float lastmics,basemics;
extern    long minclicks,maxclicks,sumclicks,thisclicks,loop1ctr;
extern    long i,n,basen;
extern    int       i0,i1,i2,i3,i4;
extern    float f0,f1,f2,f3;
extern    char quesstr;
extern    float basefit;
extern    float warn;

char s[100];

char *s0123456789 = "0123456789";
char *sa123456789 = "a123456789";
char *s12345  = "12345";
char *s123_45 = "123.45";
char *sd = "%d";
char *sdn = "%d\n";
char *sf  = "%f";
char *sf62  = "%6.2f";

void  TestInOut()
    {
        char fname[20];
        FILE     *fp;

        n = basen/1000;
        strcpy(fname,"junk");
        title("Input/Output");
        strcpy(s,"1234\n");
        fp = fopen(fname,"w");
        loop(fputs(s,fp))
        fclose(fp);
        fp = fopen(fname,"r");
        loop(fgets(s,9,fp))
        fclose(fp);
        fp = fopen(fname,"w");
        loop(fprintf(fp,sdn,i))
        fclose(fp);
        fp = fopen(fname,"r");
        loop(fscanf(fp,sd,&i1))
        fclose(fp);
        unlink(fname);
    } /* TestInOut */

void  TestString()
    {
        n = basen / 10;
        title("String functions");
        loop(strcpy(s,s0123456789) )
        loop(i1 = strcmp(s,s) )
        loop(i1 = strcmp(s,sa123456789))
    } /* TestString */

void  TestStringConversion()
    {
        n = basen / 100;
        title ("String Number Conversions");
        loop(i1 = atoi(s12345))
        loop(sscanf(s12345,sd,&i1))
        loop(sprintf(s,sd,i))
        loop(f1 = atoi(s123_45))
        loop(sscanf(s123_45,sf,&f1))
        loop(sprintf(s,sf62,123.45))
    } /*  TestStringConversion */

void  TestMath()
    {
        n = basen / 100;
        title ("Math Functions");
        loop(i1 = rand())
        f2 = 5.0;
        loop(f1 = log(f2))
        loop(f1 = exp(f2))
        loop(f1 = sin(f2))
        loop(f1 = sqrt(f2))
    } /* TestMath */
```

Листинг 4

```
/*
 * MINTIME3.C
 */

#include "mintime.h"

extern    long loop1start,loop1next;
extern    float lastmics,basemics;
extern    long minclicks,maxclicks,sumclicks,thisclicks,loop1ctr;
extern    long i,n,basen;
extern    int       i0,i1,i2,i3,i4;
extern    char quesstr;
extern    float basefit;
extern    float warn;

struct stacknode
    {
        long val;
        struct stacknode *next;
    };

typedef struct stacknode *Stackp;
Stackp stackroot;

void push (long i)
    {
        Stackp p;
        p = (Stackp) malloc(sizeof(struct stacknode));
```

```
            p->val = i;
            p->next = stackroot;
            stackroot = p;
        } // push

long pop (void)
        {
            long ix;
            ix = stackroot->val;
            stackroot = stackroot->next;
            return ix;
        } // pop

void TestMalloc()
        {
            n = basen/1000;
            title ("Malloc");
            loop(free(malloc(8)) )
            loop(push(i))
            loop(i1 = pop())
        } /* TestMalloc */
```

Листинг 5

```
# Make-файл для создания теста компиляторов

MODEL = -mm
CSRC1 = mintime1.c mintime2.c mintime3.c

result: mc7tst.tst zc3tst.tst bc2tst.tst bc3tst.tst wc8tst.tst

all: msc7tst.exe zc3tst.exe bc2tst.exe bc3tst.exe wc8tst.exe

clean:
        del *.obj

msc7tst.tst : msc7tst.exe
        msc7tst.exe 1000000 0.4 > msc7tst.tst
zc3tst.tst : zc3tst.exe
        zc3tst.exe 1000000 0.4 > zc3tst.tst
bc2tst.tst : bc2tst.exe
        bc2tst.exe 1000000 0.4 > bc2tst.tst
bc3tst.tst : bc3tst.exe
        bc3tst.exe 1000000 0.4 > bc3tst.tst
wc8tst.tst : wc8tst.exe
        wc8tst.exe 1000000 0.4 > wc8tst.tst

msc7tst.exe : mintime1.c mintime2.c mintime3.c mintime.h
    cl /AM /FPi87 /f- /G2 /Ot /Ox /Femsc7tst /Id:\windev\include;
$(CSRC1)

zc3tst.exe : mintime1.c mintime2.c mintime3.c mintime.h
    ztc $(MODEL) -f -2 -o+all -ozc3tst -Id:\ztc\include; $(CSRC1)

bc2tst.exe : mintime1.c mintime2.c mintime3.c mintime.h
    d:\bc2\bin\bcc $(MODEL) -G -f287 -2 -O -ebc2tst $(CSRC1)

bc3tst.exe : mintime1.c mintime2.c mintime3.c mintime.h
    bcc $(MODEL) -G -f287 -2 -O2 -ebc2tst -Id:\bc\include; $(CSRC1)

wc8tst.exe : mintime1.c mintime2.c mintime3.c mintime.h
    wcl $(MODEL) -ot -7 -2 -Id:\wc\h; -fe=wc8tst $(CSRC1)
```


Сегодня SPIRIT представляет

ArcView 6.3

Экономия с Удовольствием

Просмотр и Редактирование Архивов

- Распознавание архивов ARJ, LZH, ICE, ARC, PAK, ZIP, CHZ и SFX
- Быстрее и удобнее, чем оригинальный архиватор
- Распаковка, просмотр, тестирование и удаление файлов в архиве
- Работа с поддиректориями в архивном файле и самораспаковывающимися архивами

Всего 475 рублей за копию

Тысячи копий уже приносят пользу

SPIRIT
OF FREE IDEAS

Сегодня SPIRIT представляет

File Crypter 2.0

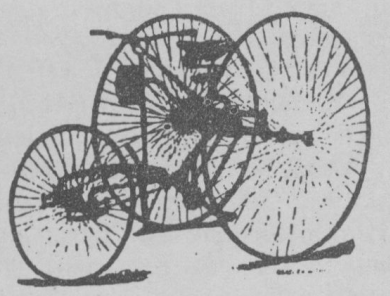
Надежнее чем Сейф

Простая система для профессиональной криптографической защиты коммерческих данных на PC

Разнообразные алгоритмы шифрования в одной среде с удобным интерфейсом :

- Надежный DES (федеральный криптостандарт США)
- Быстрый RSC (оригинальный отечественный алгоритм)
- Надежный и Быстрый ГОСТ 28147-89 (федеральный криптостандарт России)
- Цифровая подпись

Всего 4.875 рублей за копию



Особенности C++ в компиляторах Borland

А. Кельманзон

Введение

На сегодняшний день официально утвержденного стандарта C++ не существует, но, по словам президента европейского филиала Borland Франсуа Миколя на отчетном собрании BorUGA в декабре 1991, все изменения в версиях Borland C++ согласуются с AT&T Bell Laboratories и поэтому являются стандартом de-facto ([1], [2]).

Итак, речь пойдет об отличиях Borland C++ от классического C++, который описал B.Stroustrup [3].

Какие разделы языка претерпели изменения?

1.Спецификаторы доступа и наследование.
2.Перегрузка операторов (в версии 3.0).
3.Описание функций.
4.Динамическое распределение памяти.
5.Ключевое слово template (в версии 3.0).

Спецификаторы доступа и наследование

Первое,что бросается в глаза, - это появление спецификатора доступа protected, являющегося эквивалентом private, с одним отличием: данные и методы области protected "родителя" являются доступными для дочерних классов (листинг 1).

В Borland C++ 3.0 использование этого спецификатора доступа распространено и на наследование. Видимость полей родительского класса для дочерних классов представлена в таблице 1.

В последней версии Borland C++ влияние спецификаторов видимости распространилось на поля и методы типа static. Общедоступными являются только те, которые находятся в области public (в версиях C++ фирмы Borland до 3.0 поля и методы static были общедоступными, независимо от своих спецификаторов видимости). Теперь доступность полей и методов static зависит от их спецификаторов видимости:

public - общедоступные;
protected - доступны только объектам класса, в котором они описаны, объектам дочерних классов и дружественным (friend) классам и функциям;
private - доступны только объектам класса, в котором они описаны, и дружественным (friend) классам и функциям.

Таблица 1. Соответствие полей родительского класса полям дочерних классов при различных спецификаторах доступа

Поля родительского класса	Как наследуются	Поля дочернего класса	Спецификатор доступа
public	→	public	public
protected	→	protected	
private		private	
public	→	public	protected
protected	→	protected	
private		private	
public	→	public	private
protected	→	protected	
private	→	private	

Помимо нового спецификатора доступа, в Borland C++ реализовано множественное наследование, то есть у одного дочернего класса может быть несколько родительских (листинг 2).

Обратите внимание на использование ключевого слова virtual при описании классов B и C. Благодаря этому поля класса D выглядят так:

```
A::aa    199 (0xC7)
B::bb    33 (0x21)
C::cc    11 (0xB)
dd       1001 (0x3E9)
```

Компилятор определил, что класс A является общим базовым классом для B и C и поэтому наследовал поле A::aa в классе D один раз.

Перегрузка операторов

В третьей версии Borland C++ появилась возможность перегружать как префиксную, так и постфиксную форму операций инкремента и декремента (листинг 3). Это реализовано, благодаря способности объектно-ориентированного компилятора различать функции по типу передаваемых параметров.

Описание функций

Реализация C++ фирмы Borland включает в себя новое понятие ЧИСТОЙ ВИРТУАЛЬ-

НОЙ ФУНКЦИИ:

```
// Абстрактный класс AbstractBaseClass содержит описание
// методов без их реализации (virtual <тип> <метод>() = 0)
class AbstractBaseClass {
                public:
                    // Чистая виртуальная функция returnRang(),
                    // то есть обьявленная, но не реализованная
                    virtual int
                    returnRang() = 0;
                ...
};
```

Благодаря этому появилась возможность создания интерфейса использования дочерних классов через АБСТРАКТНЫЙ базовый класс, то есть, описывая чистые виртуальные функции, мы резервируем имена функций, через которые будем получать доступ к данным дочерних классов. Так как в абстрактных классах нет описания реализации самих методов, то не может существовать объектов от этих классов, а во всех дочерних классах эти методы должны быть обязательно (!) перегружены.

Наиболее очевидный пример использования абстрактных классов и чистых виртуальных функций - создание списка, в котором необходимо хранить разнородные структуры данных. Для этого каждый элемент списка описывается как указатель на абстрактный класс, а каждая структура, которую необходимо поместить в список, описывается как производная (дочерняя) от абстрактного класса или его потомка.

Появление чистых виртуальных функций - это не единственное новшество, касающееся функций-методов. Теперь они могут описываться еще с одним из двух ключевых слов: volatile или const.

```
class MyClass { public:
                void    printMyClassData( ostream& ) const { ... }
                void    setRandDate( int ) volatile { ... }
};
```

Функции типа const не могут изменять значения полей объекта и, соответственно, только эти функции могут использоваться, если объект объявлен неизменяемым (const MyClass ConstObj(...);); функции типа volatile противоположны по использованию функциям типа const, то есть могут изменять значение полей (по умолчанию все функции имеют тип volatile).

Динамическое распределение памяти

Предположим, что Вам необходимо использовать в программе динамически распределенный вектор некоторых объектов, включающих в себя также динамически распределенные данные (в нашем случае - вектор строк). Для корректной работы с памятью блоки распределенной памяти после использования должны быть освобождены. Если Вы будете использовать обыкновенный оператор delete, то будет вызван деструктор только для первого элемента вектора, а память,занимаемая остальными элементами, не будет освобождена. Для освобождения всей памяти, занятой элементами вектора, используется множественное освобождение памяти (листинг 4).

Если бы мы пользовались старой реализацией множественного оператора delete, то нам необходимо было бы указать количество объектов вектора (в нашем случае delete[5]) в деструкторе. В новой реализации это делать необязательно, т.е. достаточно написать delete[]. Дело в том, что на самом деле выделяется на 2 байта больше памяти, чем необходимо для хранения элементов вектора, а затем адрес блока сдвигается. Именно в этих 2 байтах хранится размер вектора объектов (рис. 1).Он определяется по формуле:

РАЗМЕР_ВЕКТОРА = *(((int*)НАЧАЛЬНЫЙ_АДРЕС_ВЕКТОРА)-1)

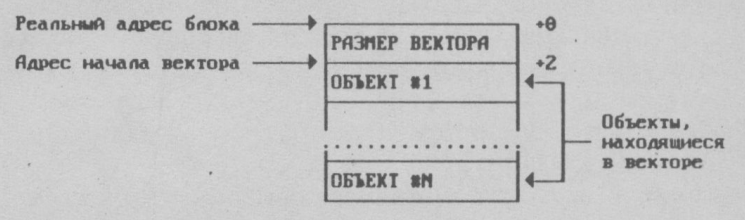


Рис. 1. Организация памяти при распределении вектора объектов

Ключевое слово template

В Borland C++ 3.0 наконец-то появилась реализация давно зарезервированного ключевого слова template. Это позволяет писать классы и функции, работающие с передаваемыми типами данных. В предыдущих версиях языка это было возможно за счет реализации средствами препроцессора обобщенных классов , что с программной точки зрения несколько сложнее и менее наглядно. В листингах 5 и 6 реализован вектор из заранее не известных элементов двумя способами.

Для каждого из объектов класса типа template компилятор генерирует свои методы, в зависимости от используемых данных, а не использует одни и те же.

Например, вызовы конструкторов на самом деле выглядят следующим образом:

```
call near ptr @@%List$ti%@$bctr$qi - List<int> x(20);
call near ptr @@%List$tf%@$bctr$qi - List<float> y(30);
```

Заключение

Помимо описанных изменений в синтаксисе языка C++, версия фирмы Borland содержит дополнительные библиотеки классов, но об этом будет рассказано в следующий раз. ■

Литература

1. Borland C++ 2.0 Programmer's Guide.
2. Borland C++ 3.0 Programmer's Guide.
3. B.Stroustrup. The C++ Programming Language.- Mass.: Addison-Wesley Reading, 1986.

Листинг 1

```
#include <iostream.h>

class ClassParent {
    public:
                ClassParent( int a, int b ) {
                        protParentDate = a;
                        privParentDate = b;
                }
    protected:
        // Доступно для объектов дочерних классов
        int     protParentDate;
        int     retPrivateDate() {
                    return privParentDate;
                }
    private:
        // Доступно только для объектов данного
        //класса и дружественных функций
        int     privParentDate;

};

class ClassChild : public ClassParent {
    public:
                ClassChild( int a, int b, int c )
                        : ClassParent( a, b )
                {
                        childDate = c;
                }
        int     resAdd() {
                        return( childDate +
                // Обращение к полю родительского
                    // класса, имеющего спецификатор доступа
                        // public или protected, возможно напрямую
                            protParentDate  +
                        // Обращение к полю со спецификатором
                    // доступа private возможно только через
                // функцию, доступную  для дочернего
            // класса, то есть имеющую спецификатор
        // доступа public или protected
                                retPrivateDate()
                            );
                }
    private:
        int childDate;
};


void main() {
    ClassChild ChildObj( 2, 5, 9 );
        cout < < "\nРезультат: " < < ChildObj.resAdd();
};
```

Листинг 2

```
// Базовый родительский класс A
class A {
    int aa;
public:
        A( int r ) { aa = r; }
};

// B и C - дочерние классы от A
class B : virtual public A {
    int bb;
public:
        B( int r ) : A(10) { bb = r; }
};

class C : virtual public A {
    int cc;
public:
        C( int r ) : A(22) { cc = r; }
};

// D - дочерний класс от B и C
class D : public B, public C {
    int dd;
public:
        D( int r ) : C(11), B(33), A(199) { dd = r; }
};

main() {
D ObjD(1001); // Инициализация объекта класса D ObjD
    // Если Вы посмотрите во встроенном отладчике после инициализации
    // (с помощью Inspect (Alt-F4)), какие поля имеет объект ObjD, то
    // увидите, что поле A::aa наследуется только один раз. Это произошло
// благодаря использованию virtual.
}
```

Листинг 3

```
#include <iostream.h>

class Oper {
    public:
            Oper(void) : x(0), status(0) {};
            ~Oper() { x = 0; status = 0; }
    Oper    &operator + +();            // Префиксная форма
    Oper    &operator + +(int);         // Постфиксная форма
    void Set( int r ) { x = r; status = 0; }
    // Функция корректирует поле x, в зависимости от
    // значения поля status, и возвращает значение поля x
    int     Ret( void ) { xSet(); return x; }

    private:
        int     x;
        int     status;         // 0 - Oper::x не изменять
                                // 1 - Oper::x должно быть изменено
                                //     перед использованием
        // Изменение Oper::x, в зависимости от статуса
        void xSet() {
                if( status ) {
                        + +x;
                        status = 0;
                }
        }
};

Oper& Oper::operator + +() {
    cout < < "Префиксная форма\n";
    xSet();
    x+ +;
    return *this;
}

Oper& Oper::operator + +(int) {
    cout < < "Постфиксная форма\n";
    xSet();
    status = 1;
    return *this;
}

void main() {
    Oper a;
    a.Set( 8 );
    + +a;           // Изменяется значение a.Oper::x
    a+ +;           // Изменяется статус a.Oper::status
    cout < < "Oper::x = " < < a.Ret();
}
```

Листинг 4

```
#include <iostream.h>
#include <string.h>
#include <stdio.h>

class MyString {
    public:
                MyString() : longStr(0), Str(NULL) {}
                MyString( char *cur_sts ) {
                        InitStr( cur_sts );
                }
            void InitStr( char *cur_str ) {
                // Инициализация объекта класса MyString
                        longStr = strlen( cur_str );
                        Str = new char[ longStr ];
                        strcpy( Str, cur_str );
                }
```

```
                ~MyString() {
                        longStr = 0;
                        delete Str;
                }
        char    *retStr() const { return Str; }

    private:
        int     longStr;  // Длина строки
        char    *Str;     // Указатель на строку
};

class MyVectorStr {
    public:
                        MyVectorStr( int size_v ) {
                        // Распределение памяти под
                    // вектор Vec, состоящий из size_v
                // объектов типа MyString
                            Vec = new MyString[ size_v ];
                        }
                        ~MyVectorStr() {
                            // Множественное освобождение
                        // памяти, занятой вектором и
                    // его элементами
                            delete []Vec;
                        }
        MyString &operator[]( int i ) {
                        // Доступ к i-му элементу вектора
                            return Vec[i];
                        }
        int             retVectorSize() {
                        // Размер вектора объектов
                            return *(((int*)this->Vec)-1);
                        }
    private:
        MyString *Vec; // Вектор объектов типа MyString
};

void main() {
    MyVectorStr   VectorStr( 5 );
    char      buf[10] = "";

        cout << "\nРазмер вектора: "
             << VectorStr.retVectorSize();
        for( int i = 0; i < 5; i++ ) {
            // Инициализация элементов вектора
            sprintf( buf, "Строка#%d", i+1 );
            VectorStr[i].InitStr( buf );
        }
        for( i = 0; i < 5; i++ )
        // Вывод на экран содержащихся в векторе объектов
            cout << "\n" << VectorStr[i].retStr();
};
```

Листинг 5. Реализация вектора с использованием обобщенных классов

```
#include <iostream.h>
#include <iomanip.h>

#define ClassArrayName(TypeEl,N)        ObArray_##TypeEl##N

#define arrayDeclare(TypeEl,N) \
class ClassArrayName(TypeEl,N) {                                \
    public:                                                     \
            ClassArrayName(TypeEl,N)( int lArray ) {            \
                /* Распределение памяти под                     \
                * вектор, состоящий из lArray                  \
                * элементов типа TypeEl                        \
                */                                              \
                arrayElem = new TypeEl[ lArray ];               \
                /* Инициализация размера вектора */             \
                sizeArray = lArray;                             \
            }                                                   \
            ~ClassArrayName(TypeEl,N)() {                       \
                /* Освобождение памяти,                         \
                * занятой под вектор                           \
                */                                              \
                / * (!) ------------------------                \
              Используется множественный оператор               \
              delete. Это необходимо для корректной             \
              работы с памятью в случае создания                \
                        вектора из объектов.                    \
                        ------------------------------- */     \
                    delete []arrayElem; arrayElem = NULL;       \
                    sizeArray = 0;                              \
                }                                               \
                /* Функция возвращает размер вектора */         \
            int     retSize() const { return sizeArray; }       \
                /* Функция доступа к i-му элементу вектора */   \
            TypeEl &operator[]( int i ) { return arrayElem[i]; } \
        private:                                                \
            TypeEl *arrayElem;      /* Вектор объектов типа TypeEl  */ \
            int     sizeArray;      /* Размер вектора               */ \
}

void main() {
                        // Вектор из 10 элементов типа int
    arrayDeclare(int,1)     MyArrayI( 10 );
                        // Вектор из 10 элементов типа float
    arrayDeclare(float,1)   MyArrayF( 10 );
        for( int i = 0; i < MyArrayI.retSize(); i++ )
                MyArrayI[i] = i*i;
        for( i = 0; i < MyArrayI.retSize(); i++ )
                cout << "\n#" << setw(2) << i
                     << " " << MyArrayI[i];
        for( i = 0; i < MyArrayF.retSize(); i++ )
                MyArrayF[i] = i*i*0.2;
        for( i = 0; i < MyArrayF.retSize(); i++ )
                cout << "\n#" << setw(2) << i
                     << " " << MyArrayF[i];
}
```

Листинг 6. Реализация вектора с использованием ключевого слова template

```
#include <iostream.h>

template <class TypeEl> class List {
    public:
                List(int);
                ~List() {
                        // Освобождение памяти,
                        //занятой под вектор
                        delete []arrayElem; arrayElem = NULL;
                        sizeArray = 0;
                }
                // Функция возвращает размер вектора
        int     retSize() const { return sizeArray; }
        TypeEl  &operator[](int);

    private:
        TypeEl  *arrayElem; // Вектор объектов типа TypeEl
        int     sizeArray; // Размер вектора
};

template <class TypeEl> TypeEl& List<TypeEl>::operator[](int i)
{
        // Доступ к i-му элементу вектора
        return arrayElem[i];
}

template <class TypeEl> List<TypeEl>::List(int n) {
        // Распределение памяти под
        // вектор из n элементов типа TypeEl
        arrayElem = new TypeEl[n];
        // Инициализация размера вектора
        sizeArray = n;
}

void main() {
    List<int>     x(5);     // Вектор из 5 элементов типа int
    List<float>   y(10);    // Вектор из 10 элементов типа int

        for( int i = 0; i < x.retSize(); i++ )
                x[i] = i*i;
        for( i = 0; i < x.retSize(); i++ )
                cout << '#' << i << '' << x[i] << '\n';
        for( i = 0; i < y.retSize(); i++ )
                y[i] = i*i*0.1;
        for( i = 0; i < y.retSize(); i++ )
                cout << '#' << i << '' << y[i] << '\n';
}
```

Итерация против рекурсии: проблема выбора

М.Трофимов

Рекурсия - очень мощное и полезное средство программирования, что было недавно продемонстрировано в ряде классических примеров на страницах журнала "Монитор": чем мощнее средство, тем нежелательнее злоупотребление им, поэтому автор статьи [1] предостерегает против бездумного использования рекурсии там, где достаточно более простой конструкции - повторения (цикла, итерации).

Другой, противоположной, но не менее нежелательной крайностью являются попытки всеми силами избежать рекурсии. У многих программистов эта привычка связана с практикой работы на архаичных языках программирования Фортран и Бейсик, в которых рекурсия была запрещена[1]. Поскольку любое рекурсивное решение всегда может быть преобразовано в итерационное (что, собственно, и делает компилятор с Вашей рекурсивной программой), каких-либо теоретических ограничений для ненавистников рекурсии не существует. На практике же попытки заменить наглядное рекуроивное решение итерационным зачастую оборачиваются неимоверным усложнением исходного текста. Попробуйте, например, переписать[2] хорошо известную программу (листинг 1, [2]) преобразования арифметических (алгебраических) выражений в постфиксную форму (бесскобочная запись). Программа работает следующим образом.

На вход (файл input) поступает выражение, записанное в привычной нам инфиксной форме:

ПервоеЧисло Операция ВтороеЧисло,

например,

a + b

На выходе будет выражение, переписанное в бесскобочной или, как ее еще называют, польской записи, где операция записывается последней:

ПервоеЧислоВтороеЧислоОперация,

то есть программа напечатает:

ab+

Закончив ввод выражения, нажмите клавишу "Ввод". Для выхода из программы введите выражение, состоящее из точки. Сеанс работы с программой может быть таким (строки на экране чередуются: введенное выражение - выдача программы ...):

```
(a+b)*(c-d)
ab+cd-*
a+b*c-d
abc*+d-
(a+b)*c-d
ab+c*d-
```

Обратите внимание на то, что в программе присутствует более сложный тип рекурсии, чем просто вызывающая сама себя процедура, то есть сначала из тела программы следует вызов процедуры expression, эта процедура, в свою очередь, вызывает процедуру term, term вызывает factor, а factor опять expression - круг замкнулся. Схематически это можно представить следующим образом:

... —> expression —> term —> factor —> expression —>...

или, в переводе на русский,

...—> выражение —> слагаемое —> множитель —>...

Систему инфиксной записи можно представить в виде следующей рекурсивной схемы:

выражение —> слагаемое [+! - слагаемое]
слагаемое —> множитель [* множитель]
множитель —> буква! (выражение),

где квадратные скобки обозначают, что заключенная в них конструкция может отсутствовать; восклицательный знак - логическую связку "или", то есть последняя строка нашей схемы читается как "множитель - это буква или выражение, заключенное в круглые скобки". Соответствие между схемами и кодом ясно из комментариев к программе.

Благодаря тому, что процедуры вложены друг в друга: factor в term, а term в expression, оказывается возможным обойтись без предварительного описания какой-либо из этих процедур с помощью директивы forward. Попробуйте расположить процедуры на одном уровне, друг за другом:

```
...
procedure factor;
begin ... end; {factor}
procedure term;
begin... end; {term}
procedure expression;
begin ... end; {expression}
...
```

Вам понадобится перед процедурой factor добавить строку

```
procedure expression; forward;
```

Иначе компилятор, встретив вызов expression в процедуре factor, выдаст ошибку "неопределенное имя". Это очень важный момент, демонстрирующий хороший и плохой стили программирования на Паскале, с точки зрения принятых в нем правил видимости имен. Иначе говоря, возможность языка Паскаль, допускающая многоуровневое вложение процедур друг в друга, чего, к сожалению,

лишены многие другие языки программирования, - вовсе не "архитектурное излишество", а мощное средство, позволяющее, в частности, писать элегантные, удобочитаемые программы для случаев сложной рекурсии. Это средство органически сочетается с требованием Паскаля об описании идентификаторов до их употребления.

Возвращаясь к программе postfix (листинг 1), отметим, что преобразование в польскую запись широко используется в трансляторах языков высокого уровня. Многие трансляторы написаны на том же языке программирования, с которого они осуществляют компиляцию, например, транслятор с языка Паскаль был реализован на языке Паскаль, а процедуры обработки выражений представлены в рекурсивной форме, подобно рассмотренной здесь программе. (Речь идет о коммерческих профессиональных продуктах, например, широко известном компиляторе PASCAL 8000/2.0 для IBM 360/370).

"Конечно, - скажут некоторые программисты, - рекурсивное решение кажется проще. Однако я реализую трудоемкий по времени счета, согласно теоретическим оценкам, алгоритм и хочу сделать эффективную программу. Поэтому лучше я сразу потрачу дополнительные усилия на написание нерекурсивной версии". К сожалению, выигрыш от таких благих намерений зачастую оказывается несоизмеримым с трудозатратами. В некоторых же случаях вообще не удается найти достаточно эффективного нерекурсивного решения. Так, в своей практической работе [3] я столкнулся с процедурой, реализующей алгоритм экспоненциальной (по времени счета) трудоемкости, причем оптимизация этой процедуры дала выигрыш времени выполнения в 10-40%, по сравнению с другими известными из литературы решениями. Интересно, что найденное в процессе оптимизации решение явилось рекурсивным, в то время как в более медленных решениях рекурсия не используется. Конечно, скорость достигается совсем не за счет рекурсии, а, если хотите, вопреки ей. Дело в том, что в современных компиляторах механизмы рекурсивного вызова реализуются настолько эффективно, что реальные потери времени счета оказываются, как правило, незначительными. Хорошей иллюстрацией могут служить программы быстрой сортировки. Это тот случай, когда переход от рекурсивной версии (листинг 2) к итерационной (листинги 3 и 4) не влечет за собой столь большого усложнения программы, как в предыдущем случае.

Процедура быстрой сортировки наиболее естественно представляется в рекурсивном виде. Однако предубеждение против рекурсии настолько сильно, что из учебника в учебник вместе с рекурсивным вариантом быстрой сортировки, как тень, следует ее нерекурсивная версия. Тень эта сопровождается словами, что, поскольку компилятор все равно преобразует рекурсию в итерацию, то гораздо большей эффективности можно достичь с нерекурсивной версией. Вот характерный результат сравнения работы разных версий (табл. 1).

Таблица 1

Быстрая сортировка - рекурсивная реализация (procedure RecQuickSort, листинг 2)	165.93 сек.
Нерекурсивная версия 1 (procedure NRQuickSort1, листинг 3)	156.48 сек.
Нерекурсивная версия 2 (procedure NRQuickSort2, листинг 4)	174.01 сек.

Выигрыш для первой нерекурсивной версии более чем скромный. И это при том, что речь идет о решениях, считающихся классикой программирования. Так, источником первых двух решений послужила широко известная книга Н. Вирта [4] (листинги 2 и 3 - несколько измененные варианты приводимых там текстов на языке Паскаль). Там же, кстати, можно найти и детальное описание алгоритма быстрой сортировки, поэтому здесь мы его не приводим.

Со второй нерекурсивной версией (листинг 4) дело обстоит еще интереснее. Ее прототипом послужила процедура из книги [5] - очень представительного сборника наиболее известных классических алгоритмов, реализованных параллельно на различных турбо языках[3]. В процедуре применен ряд "турбо трюков", так, например, она "обманывает" компилятор и работоспособна только при отключенном тестировании на соответствие индекса интервальному типу индексов массива, отведенного под стек (опция компилятора $R-); в конец тела процедуры нам пришлось добавить вызов FreeMem, в противном случае тест не мог бы закончиться из-за переполнения памяти ПК. Увы! Из таблицы следует, что все эти ухищрения только замедляют работу программы, хотя и позволяют работать с большими массивами, - экономия одного ресурса оборачивается затратой другого, и наоборот...

А вот результаты сравнения на трех выборках исходных данных для аналогичных процедур, реализованных на языке Модула-2 для ЭВМ типа Lilith (табл. 2), взятые из книги [6].

Таблица 2

Быстрая сортировка	1 выборка	2 выборка	3 выборка
Рекурсивная реализация	0.72 сек.	1.22 сек.	0.76 сек.
Нерекурсивная версия	0.72 сек.	1.32 сек.	0.80 сек.

Так в чем же выигрыш нерекурсивной версии быстрой сортировки?

Безусловно, приведенные данные недостаточны для вынесения окончательного приговора прежде всего потому, что результаты такого сравнения зависят от многих причин[4], в том числе и от частичной упорядоченности входных данных, но они заставляют задуматься и, может быть, послужат кому-то основой для дальнейших экспериментов.

Наверное, самым ярким примером неправильного применения рекурсии является рекурсивное вычисление факториала. Однако даже и в этом крайнем случае применение рекурсии приводит к очень незначительному проигрышу по времени, по сравнению с нерекурсивной версией [7].

Итак, компилятор в ряде случаев способен лучше управиться с рекурсией, чем программист со стеком, организованным в явном виде средствами языка высокого уровня. Во всяком случае, компилятор сделает эту работу гораздо быстрее. А значит, да здравствует рекурсия! Но, как отмечалось [1], пользоваться ею надо грамотно. Взять, к примеру, листинг 5, где представлена программа, иллюстрирующая некоторые хорошо известные принципы работы с бинарным деревом (подробнее см.[8]). Программа читает из входного файла input целые числа и ищет повторяющиеся по абсолютной величине, для этого строится бинарное дерево (рекурсивная процедура add). В случае если ни один из узлов дерева не содержит поля dat, равного по модулю прочитанному числу, к дереву добавляется новый узел с полем dat, равным абсолютной величине этого числа, при этом выводится сообщение, содержащее характеристики этого узла: номер n0, расстояние от корня дерева dp, значение dat. В случае если такой узел уже существует, поле индекса повтора n этого узла увеличивается на единицу и выводится соответствующее сообщение. Построение дерева заканчивается после ввода числа 0, затем программа распечатывает таблицу введенных чисел, отсортированных в возрастающем порядке (рекурсивная процедура print), и вычисляет среднее арифметическое абсолютных величин неповторяющихся чисел.

В предлагаемом варианте процедура add является примером неудачной декомпозиции. Очевидно, прежде чем прибегать к рекурсии, надо подготовить данные, вычислив абсолютное значение один раз, а не в каждом рекурсивном вызове. Кроме того, достаточно объемные фрагменты генерации новой вершины и обработки повтора в целях экономии памяти следует оформить в виде отдельных процедур, так как во время работы код каждого нового экземпляра вызываемой процедуры порождается и удаляется динамически, а указанные фрагменты нужны далеко не всем экземплярам. Все эти замечания учтены в программном фрагменте (листинг 6), который может быть вставлен в программу (листинг 5) вместо процедуры add. Хорошим упражнением на стиль программирования будет также задание организовать распечатку чисел не только в возрастающем, но и в убывающем порядке (по выбору пользователя).

Написать ли для этого новую процедуру print2 или обойтись одной, добавив в ее заголовок еще один параметр

revers : Boolean

и модифицировав соответствующим образом саму процедуру? Эту задачу предложим для самостоятельного решения.

Отметим, что структура данных бинарного дерева описывается рекурсивно, и поэтому представляется наиболее естественным обрабатывать дерево рекурсивными же процедурами. Однако далеко не всегда рекурсивные структуры данных требуют применения рекурсивных процедур. Исключив, к примеру, поле left из записи node, вы получите структуру данных, представляющую список. В этом случае нерекурсивная реализация процедур add и print (добавление нового элемента в конец списка и печать списка) становится предпочтительнее.

Интересные возможности представляет объектно-ориентированный Турбо Паскаль для порождения рекурсивных экземпляров объекта. Для иллюстрации перепишем программу postfix в виде, представленном в листинге 7; обратите внимание на отношения наследования между объектами term, factor и expression[5].

В заключение отметим, что приводимые здесь взгляды не претендуют на статус бесспорных истин, по мнению автора, в программировании вообще очень мало очевидного и бесспорного. Возможно, это наиболее сильно отличает программирование для компьютеров от других видов деятельности: точных наук и инженерного проектирования, - и заставляет снова и снова задаваться вопросом: "Что же такое программирование - наука или искусство?" ■

Литература

1. Д. Бройтман. Этюд 2. Рекурсия или не стреляйте из пушки по воробьям // Монитор.- 1992.- N 2.- С.46-48.
2. К. Йенсен, Н. Вирт. Паскаль. Руководство для пользователя и описание языка.- М.: Финансы и статистика, 1982.- С.64-65.
3. M.Trofimov. An Optimization of Procedure for Calculation of Hosoya's Index // Journal of Mathematical Chemistry.- 1991.- N8.- P.327-332.
4. Н.Вирт. Алгоритмы + структуры данных = программы.- М.: Мир, 1985.- С.96-108.
5. K. Weiskamp, N.Shammas, R.Pronk. Turbo Algorithms. A programmer's Reference.- J.Wiley & Sons, Inc., 1989.- P.42-44.
6. Н.Вирт. Алгоритмы и структуры данных.- М.: Мир, 1989.- С.126.
7. M.Trofimov. How Much Does a Recursion Cost? // Journal of Pascal, Ada & Modula-2.- 1990.- N 3.- P.6.
8. П.Грогоно. Программирование на языке Паскаль.- М.: Мир, 1982.- С.264-271.
9. С. Гудман, С. Хидетниеми. Введение в разработку и анализ алгоритмов.- М.: Мир, 1981.- С.263.
10. В.Н. Касьянов, В.К. Сабельфельд. Сборник заданий по практикуму на ЭВМ.- М.: Наука, 1986.- С.198-227.
11. N.Shammas. Object-Oriented Programming with Turbo Pascal.- NY, 1990.- P.57.
12. М.Черкашин. Объектно-ориентированное программирование: что это такое и для чего оно нужно // Монитор.- 1991.- N 1.- С.18-20.

[1] Здесь, в отличие от [1], мы будем использовать язык программирования Турбо Паскаль версии 5.0, кроме листинга 7, где используется версия 6.0 того же языка. И, как будет видно из дальнейшего, наш выбор не случаен: очень уж элегантно программируются различные виды рекурсии на этом популярном и выразительном языке.

[2] Для этого можно воспользоваться алгоритмом [9] или заглянуть в [10], где приводятся параллельные решения близкой задачи (упрощение логических формул) на языках Паскаль (рекурсивная версия) и ФОРТРАН-IV (нерекурсивная).

[3] Речь в книге идет о Турбо Бейсике, Турбо Паскале, Турбо Прологе и Турбо Си: с того времени, как она была написана, фирма Borland сократила этот ассортимент, и теперь в списке остались только Паскаль и Си (в том числе и Си + +).

[4] В одном из ближайших номеров мы подробнее поговорим о технике испытания программ на скорость.

[5] Подобные зацикливания в графе наследования нередко рассматриваются как еретические трюки в объектно-ориентированном программировании. В противовес этим взглядам процитируем [11], перевод автора: "...Что случится, если классу объектов нужно взаимодействовать с его подклассом? При том, что это представляется как противоречащее философии объектно-ориентированного программирования, подобная необходимость возникает в ряде приложений, таких, как моделирование химических реакций. Однако, поскольку подкласс определяется после родительского класса, вы не можете раньше сослаться на этот подкласс. Эта проблема разрешается использованием указателей".

Кстати, хорошим упражнением для читателей, желающих разобраться с этой проблемой, будет доведение примера 6 из [12] до работающей на Турбо Паскале программы.

Информация о новых вирусах в формате антивирусной программы AVSP 2.0.

AVSP предоставляет пользователю возможность непосредственного ввода информации о новых вирусах в программу. Благодаря регулярной публикации такой информации AVSP на Вашем компьютере не отстанет от времени и создателей новых вирусов. AVSP содержит ряд средств борьбы с неизвестными вирусами. ДИАЛОГ-МГУ обращается ко всем читателям с просьбой присылать информацию об обнаруженых новых вирусах. Для ввода приведенной ниже информации в Вашу программу войдите в пункт меню "Данные о файлах и вирусах", подменю "Изменение информации о вирусах". Для более подробного описания можно обратиться к документации (пункт "Ввод информации о вирусах из журналов").

Имя вируса: Tumen
Номер: 223.1-49124
Тип вируса: PROGRAM
Размер: 1663
Шаблон: 812E120000010306030005800 08BD0
Процедуры: *.COM { Opt(End:-1214) Mov(MP:-449,Beg:0,3) Tr(MP:-449) }
Справка: Создан, по словам авторов, в Тюмени в 1990 году. Содержит много текста типа "Full Moon tonight... The Time of Wolfbacks" и т.п. Предупреждает, что это только ДЕМО-версия (v0.5) вируса...

Имя вируса: MSU-3125
Номер: 224.1-27058
Тип вируса: PROGRAM
Шаблон: 51509CBB7502B9250C2BCB2EA0DE01
Процедуры: *.EXE { Opt(End:-2629) Mov(MP:54,Head:20,4) SubW(Head:10,$103) TrEXE(End:-3110) } *.COM { Opt(End:-2629) Mov(End:-2843,Beg:0,3) Tr(End:-2850) }
Справка: У зараженных EXE файлов длина увеличивается от 3110 до 3125 байтов. У COM файлов на 2850 байтов. Случайным образом заражаются файлы при входе в директорию, а также при запуске и обращении к программным файлам. Обнаружен в МГУ.

Внимание!!! Новая информация по вирусу IBM, который уже есть в базе данных AVSP:

Имя вируса: IBM
Номер: 130.2-1814
Тип вируса: PROGRAM
Размер: 547
Шаблон: 8ED88B1620022E891600018A362202
Процедуры: *.COM { Opt(End:-537) Mov(MP:526,Beg:0,3) Tr(MP:-10) }
Справка: Содержит строку "IBM".

Имя вируса: Vienna-377
Номер: 225.1-30752
Тип вируса: PROGRAM
Размер: 377
Шаблон: 5AB42A81F9CF077435BF00018BF2
Процедуры: *.COM { Opt(End:-360) Mov(MP:-5,Beg:0,3) Tr(MP:-17) }
Справка: Из семейства вирусов Vienna. Заражает все COM файлы в текущем каталоге.

Имя вируса: Microsoft II
Номер: 226.1-4647
Тип вируса: PROGRAM
Шаблон: 240000004D73446F73

Имя вируса: EXE-1670
Номер: 227.1-56826
Тип вируса: PROGRAM
Размер: 1670
Шаблон: 1E50530657569C2E
Процедуры: *.EXE { Opt(St:1330) Mov(MP:165,Head:2,22) Tr(End:-1670) }
Справка: Заражает только EXE файлы. Резидентный. Вроде бы содержит закодированный текст. У зараженныйх файлов устанавливает время 62 секунды.

По вопросам приобретения AVSP 2.0 обращайтесь в ДИАЛОГ-МГУ по телефонам (095) 939-38-90, 939-38-91. Подробная информация высылается бесплатно. Цена одного комплекта AVSP 2.0 на 28 октября 1992 г. составляет 1924 рубля (с учетом 28% налога). При покупке нескольких комплектов предоставляется скидка. Скидка для дилеров - до 60%.

Листинг 1. Программа преобразования арифметических выражений

```
program postfix;
var
    ch   : char;

procedure find;
    begin
        read (ch);
    end; { ____________________________________ find _____ }

procedure expression;
var
    op : char;

    procedure term;
        procedure factor;
            begin                       { множитель —> }
                if ch = '('
                 then
                   begin
                     find;
                     expression;        { (выражение) }
                   end
                else
                     write ( ch );      { ! буква }
                find;
            end; { ____________________________________ factor _ }

        begin { term }                  { слагаемое —> }
            factor;                     { множитель }
            while ch = '*' do           { [*        }
             begin
                find;
                factor;                                 { множитель] }
                write ('*')
             end;
        end; { ____________________________________ term ___ }

begin { expression }                    { выражение —> }
    term;                               { слагаемое    }
    while ( ch = '+') or (ch = '-') do  { [+!- }
     endbegin
        op := ch;
        find;
        term;                           { слагаемое] }
        write ( op );
     endend;
end; { ____________________________________expression___ }

begin { program }
    writeln (' Postfix. Введите выражение или "." -конец работы ');
    find;
    repeat
        expression;
    until ch = '.';
end.
```

Листинг 2. Рекурсивная версия процедуры быстрой сортировки

```
procedure RecQuicksort (startInd {номер первого элемента
                                  сортируемого массива the Array},
                        finishInd {номер последнего элемента
                                   сортируемого массива}
                        :integer);
var
    median    {медиана},
    SwapBuf  {буфер перестановки}
              :SortType {тип элемента сортируемого массива};
    left   {левый элемент},
    right {правый элемент}      :integer;

begin
    {определить медиану}
    left  := startInd;
    right := finishInd;
    median    := theArray [ (startInd + finishInd) div 2 ];

    {провести разделение}
    repeat
        while theArray [ left ] < median do {перебирать слева}
            inc ( left );
        while median < theArray[right] do {перебирать справа}
            dec ( right );
        if left < = right then {поменять местами}
         begin
            SwapBuf := theArray [ left ];
            theArray [ left ] := theArray [ right ];
            theArray [ right ] := SwapBuf;
            inc ( left );
            dec ( right );
         end;
    until right < = left;

    {сортировать левую и правую части: рекурсивные вызовы:}
    if startInd < right
    then
        RecQuickSort ( startInd, right );
    if left < finishInd
     then
        RecQuickSort ( left,  finishInd );
end; {_____________________________ RecQuickSort _____ }
```

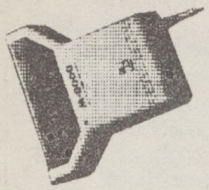
Листинг 3. Нерекурсивная процедура быстрой сортировки, версия 1

```
procedure NRQuicksort1 ( startInd, finishInd: integer);
type
    IndexRec =     record {элемент стека}
                       left, right : integer;
                   end;
    StackType = array [ 1..finish ] of IndexRec;
var
    median, SwapBuf   : SortType;
    l, r, lt, rt   : integer;
    StackHeight       : integer; {вершина стека}
    Stack             : StackType; {стек}

begin
    {инициализировать стек}
    StackHeight := 1;
    stack [1].left := 1;
    stack [1].right := finishInd;

    {основной цикл}
    repeat
        lt := stack [ StackHeight ]. left;
        rt := stack [ StackHeight ]. right;
        dec ( StackHeight );

        repeat {1-й вложенный цикл}
            l := lt;
            r := rt;
            median := theArray [ ( lt + rt ) div 2 ];

            repeat {2-й вложенный цикл}
                while theArray [ l ] < median do {перебирать слева}
                    inc ( l );
                while median < theArray [ r ] do {перебирать справа}
                    dec ( r );
                if l < = r then {поменять местами}
                 begin
                    SwapBuf := theArray [ l ];
                    theArray [ l ] := theArray [ r ];
                    theArray [ r ] := SwapBuf;
                    inc ( l );
                    dec ( r );
```

Системы и комплексы

Фирма международного концерна "НОВИНТЕХ"

ПРЕДЛАГАЕТ:



Ручные сканеры фирмы Mustek Optic GmbH с пакетами обработки изображений. Все сканеры удобны в обращении, надежны и *имеют гарантию*.

- **Matador 105** полутоновый сканер (A5, 400 dpi, 64 п/тона, эмуляция серого), цена *$315* в рублях по курсу ЦБР;
- **M-800G** серый сканер (A5, 800 dpi и 64 п/тона в п/тоновом режиме, 400 dpi и 256 серых цветов в сером режиме), цена *$588* в рублях по курсу ЦБР;
- **CG-6000** цветной сканер (A5, 200 dpi и 262 тысячи цветов в цветном режиме, 400 dpi в сером и п/тоновом режимах, высокочастотная лампа), цена *$1137* в рублях по курсу ЦБР.

Вместе со сканерами поставляются:

Scankit (Mustek GmbH) программа для сканирования, печати, чтения и записи (форматы TIFF, PCX, CUT, IMG, MSP, TGA), склейки двух полос изображения; цена пакета *входит в цену сканера*;

Rainbow Studio (VI&C Technology Inc.) Многооконный пакет (только со сканерами **M-800G, CG-6000**), сочетающий в себе утилиты сканирования, обработки изображений и графического редактора. Позволяет манипулировать яркостью, контрастностью и другими характеристиками изображений; предоставляет возможность преобразования цветного изображения в серое и полутоновое, и наоборот; цена пакета *входит в цену сканера*;

ALPHA-BIT (НПП "БИТ") пакет оптического распознавания символов. Обучаемая мультифонт программа с удобным интерфейсом, возможностью разбиения графики и последующей автокомпановкой. Пакет работает с документами форматов A5 и A4, хорошо обрабатывает "слипшиеся" буквы. Пакет "читает" как кириллицу, так и латиницу, имеет текстовый редактор. Основное достоинство пакета - минимум сложностей при работе с ним. Цена вместе со сканером *9,000 руб.*, отдельно *23,000 руб.*

Perconal Perceive OCR (Mustek GmbH) пакет оптического распознавания латиницы не требует обучения. Пакету свойственно очень высокое качество распознавания. Цена вместе со сканером *22,000 руб.*, отдельно *31,000 руб.*

Мы предлагаем и другие средства обработки текстов и изображений:

ПРОПИСЬ ("АГАМА") Программа проверки русской орфографии в среде Windows. Предлагаются варианты замены ошибочного слова. **ПРОПИСЬ** содержит 100 000 основ и "узнает" до 3 млн. словоформ.

ORFO ("Информатик") - резидентная программа контроля правописания русского текста. Обладая словарем в 250,000 основ, **ORFO** "узнает" до 3 млн. словоформ. Она также позволяет проверять согласование слов в предложении, выявляя ошибки в роде, числе, падеже и т.д. Если **ORFO** не знает какого-либо слова, Вы имеете возможность добавить его в словарь. При этом **ORFO** будет запоминать все формы нового слова. Вы можете работать с **ORFO** в любом текстовом редакторе, не выходя из него!

CopyPCX (Концерн "Новинтех") - резидентная программа, предназначенная для копирования графического изображения с экрана ПК (режимы EGA и VGA) в дисковый файл формата PCX графического редактора PC PaintBrush. **CopyPCX** позволяет выводить изображение, записанное в файле в формате PCX на экран в графическом режиме. CopyPCX практически независима от применяемых пользователем графических редакторов!

А также: мы предлагаем тестовые процессоры **Microsoft Word** ver.5.0 с русификатором **Русское Слово** ver.2.0 ("ПараГраф"); **WinWord** ver.1.1 с русификатором; локализованную русскую версию пакета **Word Perfect** ver.5.1; **шрифты для лазерных принтеров** HP LaserJet II и совместимых с ним; резидентный англо-русский и русско-английский **словарь RISS**; текстовый редактор **BRIEF** (Borland Int.); гипертекстовый семантический процессор **СЕМПРО**, позволяющий генерировать статьи и доклады, выполняющий функции персональной информационной системы.

У нас есть новейшие разработки ведущих западных фирм:

Quattro Pro 4.0 - Новая версия электронной таблицы: панель быстрого доступа к наиболее часто используемым командам; отображение информации в режиме WYSIWYG, полная поддержка мыши; средства создания интеллектуальных диаграмм (15 типов); новая версия обеспечивает поддержку **Novell**, осуществляет простой доступ к базам данных (**Paradox**, **dBASE**) и таблицам (**Lotus 1-2-3**, Symphony); таблицы и графики легко объединяются в слайд-фильмы для презентаций.

Quattro Pro for Windows - мощнейшая электронная таблица, работающая в среде **Windows**; пакет включает в себя такие "новинки" как Object Inspector, Database Desktop, Spreadsheet Notes и т.д..

В продаже также имеется **Quattro Pro @Analist** (финансовая библиотека), **Quattro Pro 4.0 Presentation** (деловая графика), русификатор **Quattro Pro**.

Borland C++ 3.1 & Application Frameworks - Компилятор, позволяющий создавать как DOS так и Windows - программы и учитывающий все особенности Windows 3.1. В комплект входят две библиотеки: Turbo Vision и Object Windows.

Borland Pascal with Objects 7.0 - это уникальная среда программирования для профессиональных разработчиков DOS и Windows-программ. Borland Pascal позволяет создавать приложения в трех режимах: DOS, DPMI и Windows. **Borland Pascal** предоставляет новую технологию разработки программ, позволяющую разделять библиотечные процедуры между DOS и Windows. Это дает возможность легко переходить от DOS к Windows путем включения специальных средств для создания DLL, используемых как в DOS, так и в Windows.

Turbo Pascal 7.0 (подмножество Borland Pascal) создает комфортабельные условия для разработки и отладки программ различной сложности в среде DOS

Paradox 4.0 мощная база данных в *десять раз* превышает быстродействие **Paradox 3.5** и намного опередила известные в настоящие время СУБД, в том числе и **FoxPro/LAN 2.0**. **Paradox 4.0** функционирует как в среде DOS, так и Windows. **Paradox 4.0** может удовлетворить самые взыскательные требования к запросам к базе данных, скорости обработки информации в сетевой среде, набору типов данных (хранение в **MEMO**-полях сложных типов данных - *графических изображений, аудио / видео информации и других двоичных данных*) и механизмам управления памятью. **Paradox** совместно с **Paradox SQL Link** предоставляет возможности для безопасного, прозрачного и высоко эффективного доступа к различным **SQL-серверам**.

Широко представленны продукты фирмы **TurboPower**: Object Professional, Object Professional C++, Async Professional, Win/Sys Library и др.

Есть и продукты фирмы **Mirosoft**: **Windows 3.1** с русификатором, **Works 2.0** - мощный интегрированный пакет.

Приобрести это Вы сможете у:

"СИСТЕМЫ и КОМПЛЕКСЫ" Москва **(095) 441-11-00.**

"ЛЭИНТЕК" (С.-Петербург) **(812) 234-89-02.**

"НОВИНТЕХ-ОПТИС" (Киев) **(044) 269-21-28.**

"НОВИНТЕХ-"ЭКСПРЕСС" (Новосибирск) **(3832) 69-75-33.**

"СОФТСЕРВИС" (Н.-Новгород) **(09222) 2-71-20.**

"НОВИНТЕХ-ВОЛГА" (Тверь) **(08222) 4-57-76.**

Приглашаем новых дилеров!

```
                end;
            until l > r; {конец 2-го вложенного цикла}

            if l < rt
             then
              begin
                inc ( StackHeight );
                stack [ StackHeight ].left := l;
                stack [ StackHeight ].right := rt;
              end;
            rt := r;
        until lt > = rt; {конец 1-го вложенного цикла}

    until StackHeight = 0; {конец основного цикла}
end; {_____ NRQuickSort _____}
```

Листинг 4. Нерекурсивная процедура быстрой сортировки, версия 2

```
procedure NRQuicksort2 ( startInd, finishInd: integer);
{$R-}
type
    IndexRec =    record
                      left, right : integer;
                  end;
    StackType = array [1..1] of IndexRec;
    StackPtr = ^StackType;
var
    median, SwapBuf   : SortType;
    l, r, lt, rt   : integer;
    stack             : StackPtr;
    StackHeight       : integer;
    RelAddr           : pointer;

begin
    GetMem ( stack, SizeOf ( IndexRec ) * (finishInd - startInd) );
    RelAddr := Stack;
    StackHeight := 1;
    stack^ [1].left := 1;
    stack^ [1].right := finishInd;
    repeat
        lt := stack^ [ StackHeight ]. left;
        rt := stack^ [ StackHeight ]. right;
        dec ( StackHeight );

        repeat
            l := lt;
            r := rt;
            median := theArray [ ( lt + rt ) div 2 ];

            repeat
                while theArray [ l ] < median do
                    inc ( l );
                while median < theArray [ r ] do
                    dec ( r );
                if l < = r then
                  begin
                    { Swapping : }
                    SwapBuf := theArray [ l ];
                    theArray [ l ] := theArray [ r ];
                    theArray [ r ] := SwapBuf;
                    inc ( l );
                    dec ( r );
                  end;
            until l > r;

            if l < rt then
              begin
                inc ( StackHeight );
                stack^ [ StackHeight ].left := l;
                stack^ [ StackHeight ].right := rt;
              end; { if }
            rt := r;
        until lt > = rt;
    until StackHeight = 0;
    FreeMem ( RelAddr, SizeOf ( IndexRec ) * (finishInd - startInd) );
end; {_____ NRQuickSort _____}
```

Листинг 5. Неудачный вариант программы работы с бинарным деревом

```
program bintree;
type
    link = ^node;           {ссылка на узел}
    node = record           {УЗЕЛ дерева:}
                left,       { левое поддерева}
                right : link;   { правое поддерево}
                dat,            { поле данных}
                no,                 { номер узла}
                n : integer;    { повторный ввод}
           end;
var
    NodeNum,        {счетчик узлов}
    same,           {счетчик повторов}
    sum,        {сумматор}
    i : integer;    {вводимое число}
    tree : link;    {дерево}

procedure add (var l: link; dt, dp: integer);
var
    absdt : integer;

begin
    absdt: = abs(dt);
    if l = nil                      { поддерево l пустое? }
    then                            { да: создать новый узел}
     begin
        new(l);
        with l^ do
         begin
            left := nil;
            right: = nil;
            dat  := absdt;
            Inc (NodeNum);
            No := NodeNum;
            N := 0;
            sum := sum + dat;
            writeln ('Новый узел # ', NodeNum, ' глубина: ', dp,' dat: ', dat);
         end
     end
    else                            { поддерево i не пустое }
     with l^ do
        if absdt < dat
         then                           {переход к левой ветви: }
            add (left, dt, dp+1) { рекурсивный вызов }
         else
         if absdt > dat
          then                          {переход к правой ветви: }
            add (right, dt, dp+1)   { рекурсивный вызов }
          else                          { dat = absdt -дублирование данных }
           begin
            Inc (n);
            Inc (Same);
            writeln ('повтор данных: ', absdt);
           end
end; {_____ add _____}

procedure print ( l : link);
begin
    if l < > nil
     then
        with l^ do
         begin
            print ( left );                 {печать левого поддерева}
            writeln ('узел#',no,' dat:',dat,' повторы:',n); {печать узла l^}
```

```
            print ( right );                {печать правого поддерева}
        end;
end; {_____________________________print_____________________}

begin
    tree := nil;          {пустое дерево}
    sum := 0;
    same := 0;
    NodeNum := 0;
    writeln (' Ввод ');
    repeat
        readln (i);
        if i < > 0
         then
            add (tree, i, 0); {построение дерева}
    until i = 0;
    print ( tree );       {печать дерева}
    writeln (' среднее арифметическое: ',sum / NodeNum:5:2,'; повторы:', same);
end.
```

Листинг 6. Фрагмент программы работы с бинарным деревом, улучшенный вариант

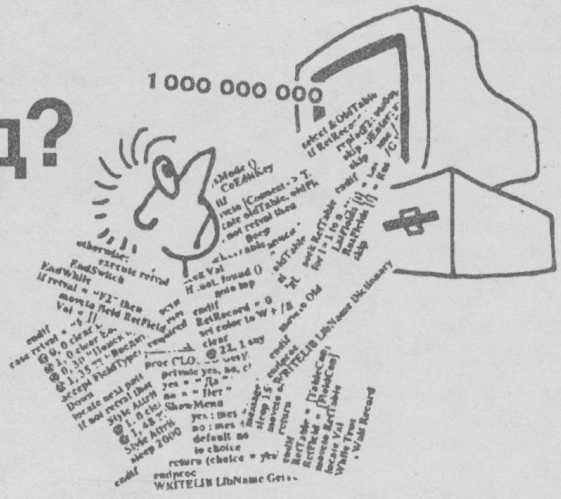
```
procedure NewNode (var l : link; dt, dp: integer);
begin
    new(l);
    with l^ do
     begin
        left := nil;
        right:= nil;
        dat := dt;
        Inc (NodeNum);
        No := NodeNum;
        N := 0;
        sum := sum + dat;
        writeln ('Новый узел # ', NodeNum, ' глубина: ', dp,' dat: ', dat);
     end
end; {_____________________________NewNode _____}

procedure SameDat (var l : link; dt, dp: integer);
begin
    with l^ do
     begin
        Inc (n);
        Inc (Same);
        writeln ('повтор данных: ', dat);
     end
end; {_____________________________SameDat _____}

procedure add (var l: link; dt, dp: integer);
begin
    if l = nil
     then                           {создать новый узел}
        NewNode (l, dt, dp)
     else
        with l^ do
            if dt < dat
             then                   {переход к левому поддереву:}
                add (left, dt, dp+1) {рекурсивный вызов}
             else
                if dt > dat
                 then               {переход к правому поддереву}
                    add (right, dt, dp+1)    {рекурсивный вызов}
                 else                {dat = absdt }
                    SameDat (l, dt, dp);
end; {_____________________________add _____}
```

Листинг 7. Программа преобразования арифметических выражений в объектно–ориентированной форме

```
program postfix;
type
    link = ^ExpressionObj;
    ident =    object
                    ch              : char;
                    procedure       find;
                    function stop   : Boolean;
               end;
    FactorObj =    object (ident )
                        e :             link;
                        procedure       factor;
                    end;
    TermObj =      object ( FactorObj )
                        procedure   term;
                    end;
    ExpressionObj =    object ( TermObj )
                            op :    char;
                            procedure   expression;
                        end;
var
    PostfixExpr : ExpressionObj;

procedure Ident.find;
begin
    read (ch)
end; { _____________________________ find _____}

function ident.stop : Boolean;
begin
    stop := ch ='.';
end; {_____________________________ stop _____}

procedure ExpressionObj.expression;
begin
    term;
    while ( ch = '+') or (ch = '-') do
     begin
        op := ch;
        find;
        term;
        write ( op );
     end;
end; { _____________________________ expression___}

procedure TermObj.term;
begin
    factor;
    while ch = '*' do
     begin
        find;
        factor;
        write ('*')
     end;
end; { _____________________________ term ____}

procedure FactorObj.factor;
begin
    if ch = '('
     then
      begin
        new (e);
        e^.find;
        e^.expression;
      end
     else
        write ( ch );
    find;
end; { _____________________________ factor ____}

begin
    writeln
    writeln (' Postfix. Введите выражение или "." -конец работы ');
    PostfixExpr.find;
    repeat
        PostfixExpr.expression;
    until PostfixExpr.stop;
end.
```

Где взять миллиард?

К. Злобин



"Уж конечно, не три миллиона старых калош!"
Н.Носов. Незнайка на Луне

Разговор у нас пойдет о суперкомпьютерах. Для начала мы попробуем уточнить смысл приставки "супер" (смысл слова "компьютер" мы предполагаем понятным) или, выражаясь более строго, попытаемся дать определение этого термина. Итак, попытка первая: "Суперкомпьютер - это ЭВМ сверхвысокой производительности". Добились ли мы чего-нибудь, кроме замены иностранных корней русскими? Да, стало понятно, что нам важна производительность ЭВМ (а не, скажем, цена). Но "сверх" - ничем не лучше, чем "супер". "Сверхвысокая" - это сколько? Миллион? Миллиард? И главное, миллиард чего? Попробуем иначе. Попытка вторая: "Суперкомпьютер - это ЭВМ, производительность которой находится на пределе технических возможностей своего времени". Уже лучше, по крайней мере, учитывается существование прогресса в этой области. Давайте поищем машину, подходящую под это определение. Ну, например,... ЭНИАК. Этот почтенный монстр состоял из 18000 электронных ламп и мог совершать более 300 умножений в секунду. И в 1946 году никакая другая машина не имела такой же производительности (никаких других просто не было). Следовательно, по нашему определению, ЭНИАК - суперкомпьютер.

Шутка? Совсем нет. Более того, в течение первых нескольких лет развития "компьютеростроения" все новые ЭВМ были "супер". Да и как иначе? Заказчиками были физики и военные (точнее, физики, работающие на военных), и за свои деньги они хотели получить возможность решать такие задачи, которые никто больше решать не умел. Положение изменилось в 1951 году, когда была выпущена на рынок (выпущена на рынок, а не изготовлена по заказу в одном экземпляре!) машина ЮНИВАК - первая ЭВМ, пригодная для обработки нечисловой информации. С тех пор производители и пользователи ЭВМ разделились на два лагеря. Одни считали, что нужно не гнаться за рекордами ради рекордов, а решать те задачи, которые ставит практика. Это благодаря им стало возможным само понятие "обычной машины". Это благодаря им в 60-е годы появилось знаменитое семейство IBM/360, в 70-е возникла идея мини-ЭВМ и воплотилась в не менее знаменитую серию PDP/11, а в 80-е на сцену вышел Его Величество Персональный Компьютер. И уж заведомо не по их вине до сих пор не воплощен в жизнь лозунг "Компьютер - в каждый дом". А в это время другие...

Другие думали точно так же. Только практика ставила перед ними иные задачи. Вообще практика - особа капризная, и задачи она перед нами ставит очень и очень разные. Для решения одних достаточно карандаша и бумаги (о чем многие пользователи персоналок в эйфории "информатизации общества" порой забывают), другие как раз по зубам вашей IBM PC, для третьих неплохо бы поднакопить деньжат на, скажем, VAX или (не смейтесь!) ЕС-1066, ну а четвертые... ну а с четвертыми непонятно, справимся ли сегодня вообще. Примеры? За ними далеко ходить не надо. Пусть мы хотим знать, как будет вести себя в воздушном потоке "Шаттл" или "Буран", или, если нам не по душе военно-космическая тематика, наш легковой автомобиль. За счет снижения сопротивления воздуха можно добиться экономии 5-10% горючего, ради этого стоит повозиться. Можно сделать из пластмассы модель нашего автомобиля (или "Шаттла") и поместить ее в аэродинамическую трубу. А можно построить математическую модель.

Обтекание предмета воздушным потоком описывается системой уравнений в частных производных, которые связывают значения и скорости изменения интересующих нас величин (например, давления воздуха) в соседних точках среды. Решить эту систему уравнений в явном виде не удается, поэтому приходится использовать численные методы. Вначале непрерывная среда заменяется на сетку, состоящую из конечного множества точек, равномерно заполняющих интересующую нас область пространства. Затем уравнения в частных производных превращаются в обычные алгебраические уравнения, связывающие значения параметров среды в соседних узлах сетки. И, наконец, получившаяся система уравнений решается каким-нибудь итерационным методом. Хлопотно, конечно, но где же обещанные принципиальные сложности?

Давайте прикинем. Пусть мы взяли сетку 100x100x100 - итого миллион точек. Пусть в каждой точке нам должны быть известны значения десяти величин. Мы получили систему из десяти миллионов уравнений. Пусть, наконец, одно уравнение содержит 100 арифметических операций, а для получения приемлемой точности нам нужно проделать 1000 итераций. Итог - десять в двенадцатой степени операций. Натурный эксперимент в аэродинамической трубе занимает около часа, хорошо бы и нам уложиться в это же время. Значит, нужно уметь совершать порядка миллиарда арифметических операций в секунду. И таких задач (их так и называют - сеточные) вокруг нас много. Это и расчет распределения температуры в атомном реакторе, и глобальное моделирование климата, и предсказание погоды на завтра. При этом возможность провести натурный эксперимент - скорее исключение, чем правило.

Расчеты сплошных сред давно уже стали классическими. В последнее время появились принципиально новые задачи, например, распознавание образов, экономическое прогнозирование, криптография. Для их решения придумано много методов, хороших и разных, которые объединяет лишь одно: все они требуют проведения громадных объемов вычислений.

Гонка за скоростью

Итак, нам нужен компьютер миллиардной производительности. А как вообще получают высокую производительность? Вспомним хорошо знакомые нам персоналки. В чем разница между IBM PC выпуска 1981 года и современной AT-386/33?

Первое, что бросается в глаза, - тактовая частота. С 4,77 МГц она возросла до 33 МГц и более. Однако здесь мы очень быстро упираемся в ограничения современной технологии (а может быть, и физики). Заставить электронные компоненты срабатывать существенно быстрее просто не удается. Даже наиболее современные "суперы" работают, как правило, на частоте 100-150 МГц - всего в 3-5 раз выше, чем у персоналок. (Кстати, разработчики супер-ЭВМ предпочитают говорить не о тактовой частоте, а об обратной величине - длительности такта, измеряя ее в наносекундах. 100 МГц - это 10 наносекунд).

Итак, заметно уменьшить длительность такта мы не можем. Значит, надо успевать сделать больше за один такт.

Вспомним опять персоналки. Старый процессор 8088 тратил на выполнение одной команды в среднем около десятка тактов, современный 80486 - менее двух. Популярные ныне RISC процессоры вышли в свет под лозунгом: "Одна команда - один такт". Как это достигается? Ведь для выполнения даже простейшей команды (например, сложения) нужно

- считать команду из памяти,
- дешифровать код операции,
- считать операнды,
- выполнить саму операцию,
- записать результат.

И все это за один такт? К счастью, этого не требуется. На помощь приходит идея конвейера. Выделенные нами стадии выполнения команды осуществляются разными частями процессора. Почему бы не заставить их работать одновременно? Представьте себе, пока одна команда считывается из памяти, предыдущая дешифруется, предшествующая ей считывает операнды и так далее.

Мы с вами пришли к схеме работы типичного RISC процессора (рис. 1) и уяснили полезный принцип: если разные части процессора могут работать одновременно - пусть они работают одновременно. Увы, сходство "супер" и "микро" на этом заканчивается. В аббревиатуру RISC входит коварное словечко "reduced" (сокращенный). Дело в том, что для выполнения любой команды нужно, помимо всего прочего,... выполнить эту команду. То есть сложить, инвертировать и тому подобное. А в красивой партитуре RISC на эту стадию отводится все тот же один такт. Вот и пришлось сократить все команды, выполнение которых в этот такт не укладывается. Даже команды умножения в типичной RISC архитектуре нет. Конечно, оставшихся команд (все логические операции, целочисленные плюс и минус и еще некоторые) для многих задач вполне достаточно. Но наши задачи вычислительные, значит, нам нужна арифметика с плавающей точкой. А каждая плавающая операция - это по меньшей мере 10-15 тактов.

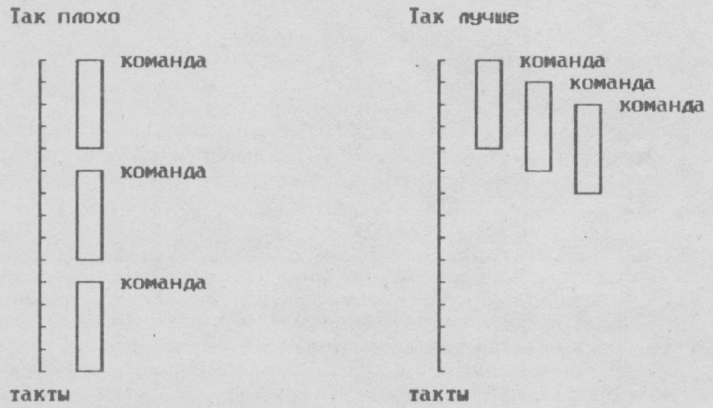


Рис. 1. Конвейер обработки команд

Что делать? Решение просто: потратим имеющийся у нас такт на запуск плавающей операции. Ответа ждать 15 тактов? А зачем его ждать? Будем выполнять следующие команды. Пусть одновременно работают арифметическое и управляющее устройства (или, как говорят практики, "арифметика" и "управление"). Однако очень скоро нам снова может встретиться арифметическая операция. А арифметика занята. Как быть? Вспомним принцип конвейера. Арифметическое устройство тоже состоит из отдельных компонент, а арифметическая операция - из отдельных стадий. Пусть результат данной операции появится на выходе устройства через 15 тактов, но в течение всего этого времени мы сможем давать на его вход новые задания, и их результаты будут появляться на выходе с той же задержкой. На практике это значит, что мы можем запускать арифметическую операцию в любой момент, когда это потребуется (рис. 2).

Следующая трудность, с которой нам предстоит столкнуться, - зависимости по данным. Да, мы можем запустить следующую операцию, не дожидаясь результатов предыдущей, но... только если эти результаты нам не нужны. А они понадобятся очень скоро. Большая часть арифметических операций входит (в терминах языков высокого уровня) в достаточно сложные выражения, и их результаты - результаты промежуточные. Пусть не на следующей команде, пусть через две-три, но нам придется остановиться и ждать результата. А всегда лучше не простаивать в ожидании чего-либо, а искать другую работу. (Это будет

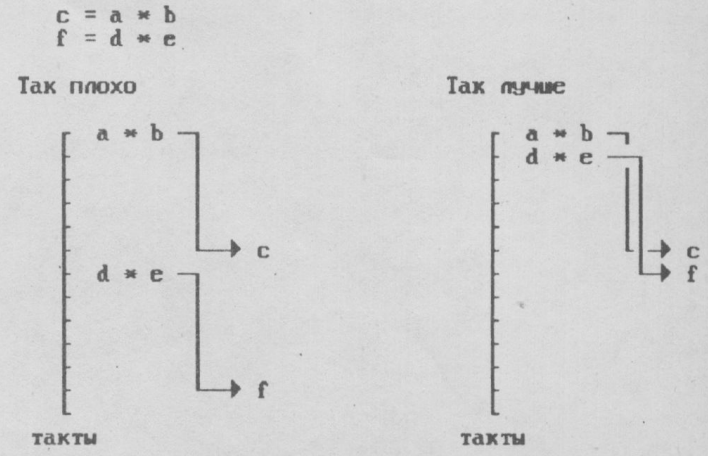


Рис. 2. Арифметический конвейер

наш второй полезный принцип). Ближайшая команда требует операнда, значение которого еще не готово? Давайте посмотрим на команду, следующую за ближайшей. Очень может быть, что ее уже можно ставить на выполнение. А если повезет, то и не одну команду можно исполнить за время ожидания. (В технической документации этот прием называется предварительной выборкой команд с переупорядочением, рис. 3).

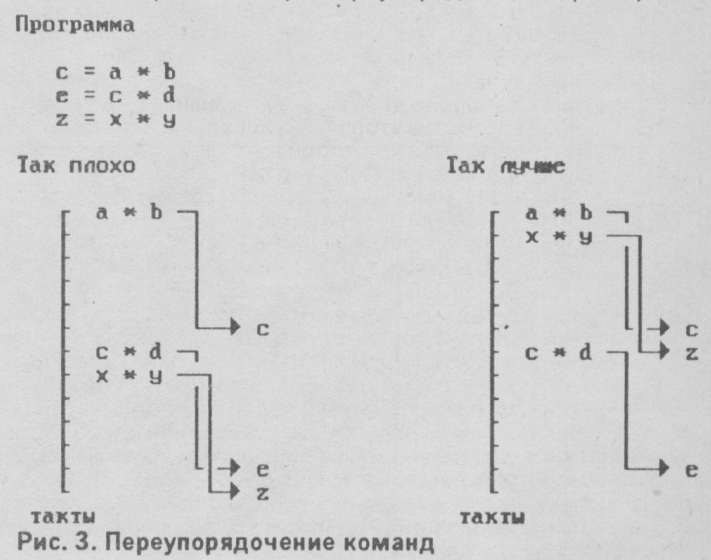


Рис. 3. Переупорядочение команд

Подведем итоги. Мы решили, что для достижения высокой прозводительности следует выполнять за один машинный такт как можно больше работы. Мы решили, что для этого нужно, во-первых, заставлять части процессора работать одновременно, а во-вторых, избегать ожидания чего-либо. Мы нашли следующие пути реализации этих принципов:

- конвейерная выборка команд,
- конвейерные арифметические устройства,
- переупорядочение команд.

Практика показывает, что совместное их использование позволяет добиться скорости 0.5-0.7 плавающих команд за такт. При длительности такта в 100 нс это дает 50-70 Mflop/s (миллионов операций с плавающей запятой в секунду). Это, конечно, много лучше, чем персоналка, но еще не обещанный миллиард. Что дальше?

VENI, VIDI, VECTOR

Разные люди используют термин "архитектура ЭВМ" для обозначения двух совершенно разных вещей. Для одних это регистры процессора, методы адресации, система команд, словом, то, с чем имеет дело программист на языке низкого уровня. Для других же - шины, магистрали, устройства, то есть все, что архитектуру, в понимании первых, реализует. То, о чем мы до сих пор говорили, касалось второго значения "слова" архитектура. С точки зрения программиста мы оставались в рамках обычной машины с архитектурой фон Неймана. Пусть на самом деле команды исполняются одновременно, пусть даже в обратном порядке, программист этого не видит. Конечно, в таком подходе есть свои преимущества: не надо переписывать про-

Листинг 1. Программа для 80X86

Исходный текст

```
for (i = 0; i < 1000; i++)
    c[i] = a[i] + b[i]
```

Код, порожденный компилятором Turbo C 2.0

```
        xor     si,si
        jmp     short @5
@4:
        mov     bx,si
        shl     bx,1
        shl     bx,1
        shl     bx,1
        lea     ax,word ptr [bp-24000]
        add     bx,ax
        FLD     qword ptr [bx]
        mov     bx,si
        shl     bx,1
        shl     bx,1
        shl     bx,1
        lea     ax,word ptr [bp-16000]
        add     bx,ax
        FADD    qword ptr [bx]
        mov     bx,si
        shl     bx,1
        shl     bx,1
        shl     bx,1
        lea     ax,word ptr [bp-8000]
        add     bx,ax
        FSTP    qword ptr [bx]
        FWAIT
        inc     si
@5:
        cmp     si,1000
        jl      @4
```

граммы, не надо переучивать программистов. Но в смысле быстродействия мы получили все, что можно. Что же мешает идти дальше? Рассмотрим пример.

Типичный фрагмент типичной вычислительной задачи - сложить тысячу пар чисел. На языке высокого уровня мы, не задумываясь, пишем что-то вроде

```
for (i = 0; i < 1000; i++)
        c[i] = a[i] + b[i];
```

А теперь посмотрите, во что это превратил компилятор (листинг 1). На одну команду плавающего сложения приходится 24 служебных операции, служащих для организации цикла. Все они относятся, строго говоря, к накладным расходам. Конечно, соотношение это нетипично и иллюстрирует, в первую очередь, что происходит при бездумном подходе к делу. (Когда я только задумал этот пример, я полагал, что на "нужную" команду придется 5-10 служебных. Результат компиляции меня ошеломил). Приличный (оптимизирующий) компилятор на приличной (хотя бы VAX) архитектуре может достичь результата 1:1 - на одну плавающую команду одна служебная - или даже чуть лучше. Но это уже предел. Чтобы двигаться дальше, нужно научиться обходиться без накладных расходов.

В больших вычислительных задачах, как правило, требуется выполнять одни и те же действия над большими массивами чисел. Сложение двух векторов - случай очень характерный. Возникает простая идея: давайте введем такую операцию в систему команд. Назовем ее, естественно, векторным сложением. Укажем ей, как и для обычной операции, адреса операндов, результата, дополним их указанием длины векторов (количества элементов в них) и - поехали. Пусть устройство управления в течение каждого такта само выбирает по паре чисел из векторных операндов, посылает их в арифметическое устройство (которое, разумеется, будет конвейерным), принимает из него сумму и посылает ее в соответствующее место вектора результата.

Красота! Каждый такт мы получаем результат плавающей операции, накладные расходы равны нулю. Но все же давайте внесем еще одно усовершенствование. Воспользуемся нашей старой идеей: не будем ждать завершения работы команды. Для этого придется разделить устройство управления и устройство организации неявного цикла для векторных команд. Теперь мы можем запустить векторную операцию и, пока она занимается суммированием десятков или тысяч пар чисел, выполнять следующую команду. Например... опять векторную. Дело в том, что в машине может быть несколько арифметических устройств. Обычно они специализированы: одно выполняет сложение, другое - умножение, третье - логические операции и так далее. И если наша векторная команда полностью загрузила сложение, это не значит, что устройство умножения должно простаивать. А самый эффективный способ дать ему работу - другая векторная команда. В идеале мы можем выполнять каждый такт столько команд, сколько у нас есть арифметических устройств. А никто не мешает нам установить в машине два или три устройства сложения, столько же устройств для умножения и так далее.

Конечно, далеко не все используемые в вычислительных задачах алгоритмы сводятся к почленным операциям над векторами. Однако можно выписать с десяток типичных "элементарных" циклов, которые уже охватывают большую часть потребностей, возникающих при вычислениях, и для каждого из них ввести специальную команду. Это усложнит архитектуру, но зато позволит векторизовать (а значит, сделать более эффективными) многие распространенные алгоритмы. Эти элементарные циклы вынесены в таблицу 1. Попробуйте понять, к каким из них сводится, например, алгоритм перемножения матриц.

Таблица 1. Циклы и векторные операции

Типичный цикл	Векторная конструкция
`for (i = 0; i < n; i++)` `  cc[i,k] = aa[i,k] + bb[i,k];`	Векторы с шагом > 1 (несмежные)
`for (i = 0; i < n; i++)` `  d[i] = a[i] + b[i] * c[i];`	Сцепленные операции

`s = 0;`
`for (i = 0; i < n; i++)`
`  s = s + a[i];`
	Продольная операция (редукция)

`for (i = 0; i < n; i++)` `  c[i] = mask[i] ? a[i] : b[i]`	Слияние по маске (вектору условий)

`j = 0;`
`for (i = 0; i < n; i++)`
`  if (mask[i]) b[j++] = a[i];`
	Сборка по маске (вектору условий)

`j = 0;`
`for (i = 0; i < n; i++)`
`  if(mask[i]) b[i] = a[j++];`
	Разборка по маске (вектору условий)

`for (i = 0; i < n; i++)` `  b[i] = a[index[i]];`	Сборка по индексу (вектору смещений)
`for (i = 0; i < n; i++)` `  a[index[i]] = b[i];`	Разборка по индексу (вектору смещений)

Собственно, в идее заменить цикл часто встречающегося вида специальной командой ничего особенного нет. Вспомните хотя бы строковые команды процессоров семейства 80X86. Но все преимущества такой команды раскрываются лишь в соответствующем окружении - в машине с конвейерным выполнением операций. Эти машины так и называют - ЭВМ векторно-конвейерного типа, к нему относится большая часть современных суперкомпьютеров.

Сходим на рынок

Итак, мы решили, что нам нужен мощный векторно-конвейерный компьютер. Поскольку нужен он, по видимому, не только нам, можно надеяться, что "там" спрос давно уже породил соответствующее предложение. Сразу же вспоминается Cray. В конце 70-х годов фотографии этого "самого дорогого в мире диванчика" обошли страницы всех (и не только компьютерных) журналов. Фактически именно тогда получила распространение идея суперкомпьютеров как особого класса вычислительной техники, а слово "крэй" стало жаргонным наименованием любого его представителя. Это была машина-легенда, машина-эпоха. Впрочем, почему "была"? Модель Cray-1, хотя и снята с производства в 1983 году, продолжает трудиться в ВЦ крупных фирм и университетов всего мира (только не у нас), помогая Джорджу Лукасу снимать "Звездные войны", а НАСА вести настоящие космические корабли.

Тем временем Сеймур Крэй выпускает новые, еще более быстрые машины, продолжая называть их своим именем. И не он один. Хотя Control Data Corporation, основной его конкурент лет десять назад, сошла с дистанции, ее место заняли вездесущие японцы - Fujitsu, Hitachi, NEC... Не отстает и Европа в лице германской Siemens, да и в США не сдаются Amdal, Convex и, конечно, IBM. К сожалению, далеко не по всем моделям нам доступна подробная информация, но рекламные цифры производительности привести можно (табл. 2).

Таблица 2. Современные суперкомпьютеры

Фирма	Модель	Максимальное число процессоров	Скорость, Mflop/s	Выпущено на 1988 г.
Cray Research	Cray X-MP	4	1200	123
	Cray 2	4	1800	11
	Cray Y-MP	8	3600	-
	Cray 3	16	18000	-
ETA systems	ETA-10G	2	750	1
	ETA-10E	2	950	-
	ETA-10Q	8	6800	2
	ETA-10P	8	10000	-
Fujitsu	VP-30E	1	220	16
	VP-50E	1	280	16
	VP-100E	1	475	10
	VP-200E	1	850	4
	VP-400E	1	1700	2
Hitachi	S-810/5	1	160	вместе - около 20
	S-810/10	1	315	
	S-810/20	1	630	
	S-820/60	1	1500	
	S-820/80	1	3000	
IBM	IBM 3090/NF	6	600	около 200
NEC	SX-JA	1	250	вместе - около 20
	SX-IEA	1	330	
	SX-IA	1	665	
	SX-2A	1	300	

Нужны суперкомпьютеры? Выбирайте! Вот только... Cray-2, например, стоит около 20 миллионов долларов. Остальные представленные здесь модели имеют цену того же порядка. Кстати, новый компьютер, относящийся к классу "супер", всегда стоит примерно такую сумму. Видимо, те, кому нужна предельная производительность (вспомним определение), готовы примерно столько за нее платить, независимо от того, чему этот предел в данное время равен. А сам предел растет по хрестоматийному экспоненциальному закону. Начав с 300 операций в 1946 году, он удваивался каждые два года и на начало 90-х составил тот самый миллиард операций в секунду (рис. 4).

Итак, берем 20 миллионов долларов ... Что? Как раз сегодня у нас нет 20 миллионов долларов? Ну что ж, посмотрим, что можно найти за рубли. Тем, кто привык к "цельностянутым" ЕС и СМ ЭВМ, трудно в это поверить, но у нас есть фирмы, которые без громкой рекламы давно и профессионально создают суперкомпьютеры. Одна из них - Институт точной механики и вычислительной техники имени С.А.Лебедева.

Может быть, кто-то из читателей еще успел поработать на

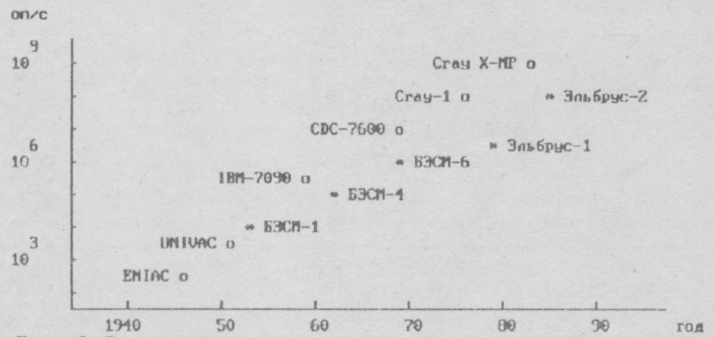


Рис. 4. Рост производительности

БЭСМ-6. Созданная в ИТМиВТ в конце 60-х годов (разработкой руководил наш замечательный ученый Сергей Алексеевич Лебедев, чье имя институт получил впоследствии), она успешно работала в различных ВЦ до середины 80-х. Несколько поколений наших программистов ласково называли ее "старушка БЭСМ". И с производительностью миллион операций в секунду она, несомненно, была для своего времени суперкомпьютером. В начале 70-х линия БЭСМ была продолжена машиной АС-6. Она, быть может, не столь известна широким массам пользователей, но зато была основной машиной, используемой Центрами управления полетами в наших космических программах. Именно АС-6 поддерживала стыковку "Союза" с "Аполлоном" в 1975 году.

В 1979 году в ИТМиВТ был создан "Эльбрус" - новая машина с оригинальной архитектурой. (Злые языки почему-то называют его Эль-Барроуз. Да, эта фирма первой применила тегированную память и безадресную (стековую) систему команд, но, следуя такой логике, любую ЭВМ можно считать скопированной с "аналитической машины" Бэббиджа. Копировать архитектуру до деталей - это одно, а пользоваться найденными кем-то хорошими идеями - это все же несколько другое). Модель "Эльбрус-2", сданная в эксплуатацию в 1985 году, имеет производительность десять миллионов операций в секунду (до ста миллионов в максимальной десятипроцессорной конфигурации). Это позволило отнести ее на момент создания к суперкомпьютерам, и до настоящего времени она является самой производительной из используемых у нас машин.

Ниже мы подробнее рассмотрим последнюю разработанную в ИТМиВТ машину, которая может стать нашим "компьютером 90-х". Ее рабочее название - модульный конвейерный процессор.

Как это делается

Давайте рассмотрим на паре примеров, как хорошие принципы применяются на деле. Начнем с ветерана Cray-1. Это была первая векторно-конвейерная машина, получившая широкую известность. Справедливости ради отметим, что у него было по крайней мере два предшественника - STAR-100 фирмы Control Data Corporation и ASC фирмы Texas Instruments, но коммерческого успеха они не имели. Краткий портрет:

- название: Cray-1,
- разработчик: Cray Research,
- год создания: 1976,
- производительность, Mflop/s: до 150,
- тактовый период, нс: 12,5,
- разрядность слова, бит: 64,
- оперативная память, слов: до 4 миллионов.

Прежде чем продолжать, мы должны признаться в небольшом жульничестве. Выше утверждалось: "В идеале мы можем выполнять в течение каждого такта столько команд, сколько у нас есть арифметических устройств". Это верно только в случае, если мы можем в течение каждого такта поставлять им нужные операнды и куда-то отправлять результаты. А пропускная способность шины между процессором и памятью, как правило, ограничена. В Cray-1 эта проблема особенно остра, так как его память способна поставлять всего одно слово за такт. Выход был найден в использовании векторных регистров. Векторный регистр по назначению подобен обычным регистрам процессора - это сверхоперативная память для хранения часто используемых величин. Только величиной здесь является вектор - 64 числа с плавающей точкой. Если алгоритм предусматривает многократное повторение операций над одними и теми же данными (а вычислительные алгоритмы очень часто бывают итерационными), отсутствие обращений к памяти приводит к заметному повышению быстродействия.

В процессоре Cray-1 имеется 12 функциональных устройств, но только три из них предназначены для выполнения векторных операций с плавающей точкой. Это сумматор, умножитель и устройство обратной аппроксимации с длиной конвейера 6, 7 и 14 тактов соответственно. (Последнее выполняет деление единицы на аргумент с "половинной" точностью. Обычная операция деления производится программно с помощью обратной аппроксимации и трех умножений. В случае векторных вычислений обратную аппроксимацию можно совместить во времени с одним из умножений, что позволяет находить частное за три такта). Максимальная производительность достигается при одновременной работе конвейеров сложения и умножения, что позволяет получать каждый такт два результата и показывать теоретическую производительность 160 Mflop/s, на практике из-за накладных расходов она снижается до 150 Mflop/s.

Второй наш пример - модульный конвейерный процессор (МКП). Краткий портрет:

- название: модульный конвейерный процессор,
- разработчик: ИТМиВТ,
- год создания: 1992 (?),
- производительность, Mflop/s: до 500,
- тактовый период, нс: 10,
- разрядность слова, бит: 64,
- оперативная память, слов: до 8 миллионов.

Процессор содержит 8 функциональных устройств (по два устройства для сложения, умножения, деления и логических операций); длина конвейера делителя - 22 такта, остальных устройств - 14 тактов. В принципе все они могут работать одновременно, но оперативная память МКП может выдавать не более шести слов за такт. Применение сцепленных операций (табл. 1) позволяет воспользоваться ими для загрузки 5 устройств и получать таким образом 5 результатов за такт (из которых 4 - промежуточные), что дает производительность 500 Mflop/s. Одновременно с этим могут работать и невекторные команды.

Кроме арифметики, процессор содержит еще два устройства управления (дело в том, что управление МКП выполняет команды не каждый такт, как в нашей идеальной схеме, а вдвое медленнее; введение двух управляющих устройств, обрабатывающих "в противофазе" два независимых потока команд, позволяет каждый такт передавать в арифметическое устройство заказ на операцию), устройство сортировки векторов (выполняет пересылку данных, а также операции сборки/разборки по маске и индексу, табл. 1), локальную память (до 8 мегаслов) и адаптер системного канала. МКП может работать в составе многопроцессорной системы, в которую входят до 32 процессоров (в типичной конфигурации 4-8), до 32 блоков массовой памяти (емкостью до 512 Мбайтов каждая, со скоростью обмена до 600 Мбайтов/с), быстрые дисковые накопители и адаптеры ввода-вывода. Все это связано средой передачи данных (системным каналом) со скоростью работы до 800 Мбайтов/с. Кроме того, в систему, как правило, входит универсальная ЭВМ для связи с внешним миром (frontal computer). Упоминание о системе с "Эльбрусом-Б" в качестве такой ЭВМ можно встретить в литературе под названием "Эльбрус 3-1".

А на самом деле?

Опытный читатель при виде фразы "у нас разработано..." инстинктивно проникается недоверием. В самом деле, что из описанного выше реально существует? В настоящее время (май 1992) имеется "в железе" четыре экземпляра МКП, находящихся на разных стадиях отладки (аппаратуру, как и программы, приходится отлаживать). На одном из них работает урезанная версия операционной системы и считаются реальные пользовательские задачи из области газодинамики. В половинной конфигурации (четыре арифметических устройства и одно устройство управления) и при тактовом периоде 14 нс достигается на отдельных тестах производительность 130 Mflop/s, на задаче обращения матрицы 100x100 - 35 Mflop/s. До конца года будет запущена полная конфигурация, что удвоит указанные цифры, снижен до 12 нс тактовый период, подключена массовая память. Первые поставки покупателям планируются на 1993 год. ■

АКЦИОНЕРНОЕ ELITSoft ОБЩЕСТВО

UPS

Устройство UPS предназначено для обеспечения бесперебойного питания ПЭВМ типа IBM PC AT/XT.

При возникновении перебоев или при полном отключении питающей сети устройство UPS осуществляет питание нагрузки с помощью автономного источника - аккумуляторной батареи.

Блок UPS обеспечит надежную защиту Вашего компьютера и сохранность программного обеспечения во время сбоя электроэнергии.

Основные характеристики:

мощность	900 ВА,
время автономной работы	40 мин.,
время переключения	5-7 мсек,
рабочий диапазон	180-240 В,
гарантия	1 год.

В конструкции предусмотрены

фильтрация сетевых помех,
звуковая и световая сигнализация,
индикация перегрузки,
индикация уровня заряда батарей,
герметичные, необслуживаемые аккумуляторные батареи.

Поставка со складов в Москве, Санкт-Петербурге, Чите, Рязани.
Любая удобная Вам форма оплаты.

TVS

Телевизионная охрана объектов внутри и вне помещений

Система TVS - это обнаружение движущихся объектов на контролируемой территории.

Высокая чувствительность, устойчивость к воздействию помех (дождь, снег, качающиеся ветки деревьев и др.) - отличительные черты TVS.

TVS способна обнаружить движущиеся объекты, горизонтальный и вертикальный размер которых составляет менее 1% от соответствующего размера изображения охраняемого объекта, а контраст не более 75%.

TVS - это комплекс, состоящий из рабочего места оператора: монитора, блока управления системой и набора видеокамер. Все видеоканалы системы работают одновременно и независимо друг от друга. Для каждого канала параметры: зона охраны произвольных размеров, чувствительность, минимальное время обнаружения объекта, - настраиваются и фиксируются оператором автономно друг от друга.

Адрес:	*105215, Москва, 11-я Парковая, д. 44, корп. 1*
Телефоны:	
секретариат	*(095) 965 3529, 532 9614*
отдел сбыта	*(095) 965 3528, 965 3530*
Факс:	*(095) 965 3530, 532 9587, 965 3529*
Телетайп:	*205211 КТУР*

Уважаемые читатели!

Мы предлагаем Вашему вниманию

компьютерный "ларек".

Мы преследуем две основные цели:

— дать возможность отечественным разработчикам инструментальных средств выйти на российский рынок программного обеспечения;

— обеспечить как можно более широкий круг читателей недорогими и оригинальными средствами разработки программного обеспечения.

Код	Продукт	Краткая характеристика	Объем поставки	Цена, $	Демо	Установщик	Требования к оборудованию	Сопровождение пользователей	Число продаж	Скидка по upgrade	Скидки в оплате	Примечания
SL-01	Turbo Games	Объектно-ориентированная библиотека для Turbo Pascal 6.0. Спрайты, звуковые эффекты, трехмерная графика, стереоэффекты. Доступ к ZIP файлам.	2 дискеты + документация	12 для индивидуальных покупателей, 25 для организаций	+	+	EGA адаптер, IBM PC/AT	+		+	+	Готовится версия для VGA
SL-02	Data Vision 1.5	Объектно-ориентированная библиотека для Turbo Pascal 6.0. Создание и управление реляционными базами данных. Поддерживаются DBF и DB форматы. Поддержка электронных таблиц.	2 дискеты + документация	12	+	—	IBM PC/XT	+	37	+	—	Версия 2.0 (осень 93 года) будет поддерживать файлы СУБД Clarion и содержать генератор отчетов
IS-01	HyperRef	Инструментальное средство для создания гипертекстовых систем. Поддержка текстовой информации, изображений и исполняемых программ.	2 дискеты по 1,2 Мб	35	+	—	IBM PC/AT, EGA/VGA, 3 Мб свободного пространства на HDD, 350 Кб ОЗУ– просмотр документа,450 Кб – редактирование документа.					
CS-01	QuickTools V1.0	Средство разработки диалога в Paradox.	1 дискета 1,2 Мб или 1,44 Мб	10	+	+	IBM PC/AT, EGA/VGA, 3 Мб на HDD, 1–2 Мб ОЗУ.		+			
UT-01	ArcView 6.1	Просмотр и редактирование заголовков архивных файлов с возможностью подключения собственных типов архивов.	1 дискета 360 Кб	1	—	+	IBM PC/AT, Hercules, EGA/VGA	+				
SE-01	Crypter 2.0	Система шифрования файлов. Поддерживаются алгоритмы DES, ГОСТ	1 дискета 360 Кб	12	—	+	IBM PC/AT, Hercules, EGA/VGA	+				
IS-02	OKO 1.0	Инструментальное средство для разработки приложений, интерфейс сходен с интерфейсом электронной таблицы. Позволяет быстро переносить документы с PC и объединять их в единую систему обработки данных. Реализовано как верхняя оболочка сетевой СУБД dbVISTA.	4 дискеты по 360 Кб или 1 дискета 1,2 Мб Документация 2 файла на установочной дискете.	18	+	+	IBM PC/XT, EGA/CGA/VGA, 1 Мб HDD, 640 Кб ОЗУ	+		+ (18 $ за 1-ю установку и 3 за каждую последующую)	+ (20% для новых версий для пользователей OKO)	
SC-01	АвтоГРАФ	Пакет для создания на экране графиков с результатами обработки числовых данных и печати их на 9– и 24–игольчатых принтерах. Поддержка форматов ЕСКД.	1 дискета 360 Кб	10	+	+	IBM PC/XT, EGA/VGA, 500 Кб на HDD, ОЗУ – 200 Кб	+	70	+	+ для учебных заведений	Готовится версия 3.0

Если Вас заинтересовали приведенные здесь продукты или Вы хотите получить дополнительную информацию, звоните, пишите, приезжайте.

Оплата принимается только в рублях по текущему курсу.

Адрес для писем: 117419 Москва, а/я 765

Тел.: (095) 238-99-33 Факс: (095) 238-32-78

Всем кто верит в компьютерные технологии

ТОО "РАКУРС"

Официальный дистрибьютер
НПЦ САПР САПСАН
предлагает **РАБОЧУЮ СТАНЦИЮ БЕСТА**

Программное обеспечение–для любых классов задач :

- инструментальные средства, СУБД,
- X–Windows, OSF Motif,
- информацинно–поисковые системы:
  - банковская деятельность,
  - АСУ "Производство",
  - АСУ "Больница",
  - АСУ "Пожарная охрана" и т.д.
- графические и технологические системы:
  - прочностный анализ,
  - моделирование поверхностей,
  - АРМ инженера–конструктора и т.д.

Современный уровень вычислительной техники:

- процессор Motorola 68020/68030,
- быстродействие – до 9 MIPS,
- тактовая частота 16/25/33 МГц,
- оперативная память 4/8/16/32/64 Мб,
- винчестерский накопитель до 1.5 Гб,
- стриммерный накопитель 155 Мб,
- графический контроллер 1280x1024, 256 цветов,
- сетевой контроллер Ethernet,
- наработка на отказ – не менее 30 000 часов.

Гарантия качества и надежности рабочей станции БЕСТА:

- современная технология :
  - контроль элементной базы,
  - автоматизированная сборка,
  - пайка волной,
  - тестирование модулей
- импортные комплектующие и периферия
- модульная компоновка "под заказчика"

Многопользовательская рабочая станция БЕСТА с операционной системой UNIX V– альтернатива сетям на базе ПК.

РАКУРС

Телефон: (095) 277–88–72